Network Working Group   J. Lang, Ed.
Request for Comments: 4204   Sonos, Inc.
Category: Standards Track   October 2005

Link Management Protocol (LMP)

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

For scalability purposes, multiple data links can be combined to form a single traffic engineering (TE) link. Furthermore, the management of TE links is not restricted to in-band messaging, but instead can be done using out-of-band techniques. This document specifies a link management protocol (LMP) that runs between a pair of nodes and is used to manage TE links. Specifically, LMP will be used to maintain control channel connectivity, verify the physical connectivity of the data links, correlate the link property information, suppress downstream alarms, and localize link failures for protection/restoration purposes in multiple kinds of networks.

Table of Contents

1. Introduction

Networks are being developed with routers, switches, crossconnects, dense wavelength division multiplexed (DWDM) systems, and add-drop multiplexors (ADMs) that use a common control plane, e.g., Generalized MPLS (GMPLS), to dynamically allocate resources and to provide network survivability using protection and restoration techniques. A pair of nodes may have thousands of interconnects, where each interconnect may consist of multiple data links when multiplexing (e.g., Frame Relay DLCIs at Layer 2, time division multiplexed (TDM) slots or wavelength division multiplexed (WDM) wavelengths at Layer 1) is used. For scalability purposes, multiple data links may be combined into a single traffic-engineering (TE) link.

To enable communication between nodes for routing, signaling, and link management, there must be a pair of IP interfaces that are mutually reachable. We call such a pair of interfaces a control channel. Note that "mutually reachable" does not imply that these two interfaces are (directly) connected by an IP link; there may be an IP network between the two. Furthermore, the interface over which the control messages are sent/received may not be the same interface over which the data flows. This document specifies a link management protocol (LMP) that runs between a pair of nodes and is used to manage TE links and verify reachability of the control channel. For the purposes of this document, such nodes are considered "LMP neighbors" or simply "neighboring nodes".

In GMPLS, the control channels between two adjacent nodes are no longer required to use the same physical medium as the data links between those nodes. For example, a control channel could use a separate virtual circuit, wavelength, fiber, Ethernet link, an IP tunnel routed over a separate management network, or a multi-hop IP network. A consequence of allowing the control channel(s) between two nodes to be logically or physically diverse from the associated data links is that the health of a control channel does not necessarily correlate to the health of the data links, and vice-versa. Therefore, a clean separation between the fate of the control channel and data links must be made. New mechanisms must be developed to manage the data links, both in terms of link provisioning and fault management.

Among the tasks that LMP accomplishes is checking that the grouping of links into TE links, as well as the properties of those links, are the same at both end points of the links -- this is called "link property correlation". Also, LMP can communicate these link properties to the IGP module, which can then announce them to other

nodes in the network. LMP can also tell the signaling module the mapping between TE links and control channels. Thus, LMP performs a valuable "glue" function in the control plane.

Note that while the existence of the control network (single or multi-hop) is necessary for enabling communication, it is by no means sufficient. For example, if the two interfaces are separated by an IP network, faults in the IP network may result in the lack of an IP path from one interface to another, and therefore an interruption of communication between the two interfaces. On the other hand, not every failure in the control network affects a given control channel, hence the need for establishing and managing control channels.

For the purposes of this document, a data link may be considered by each node that it terminates on as either a 'port' or a 'component link', depending on the multiplexing capability of the endpoint on that link; component links are multiplex capable, whereas ports are not multiplex capable. This distinction is important since the management of such links (including, for example, resource allocation, label assignment, and their physical verification) is different based on their multiplexing capability. For example, a Frame Relay switch is able to demultiplex an interface into virtual circuits based on DLCIs; similarly, a SONET crossconnect with OC-192 interfaces may be able to demultiplex the OC-192 stream into four OC-48 streams. If multiple interfaces are grouped together into a single TE link using link bundling [RFC4201], then the link resources must be identified using three levels: Link_Id, component interface Id, and label identifying virtual circuit, timeslot, etc. Resource allocation happens at the lowest level (labels), but physical connectivity happens at the component link level. As another example, consider the case where an optical switch (e.g., PXC) transparently switches OC-192 lightpaths. If multiple interfaces are once again grouped together into a single TE link, then link bundling [RFC4201] is not required and only two levels of identification are required: Link_Id and Port_Id. In this case, both resource allocation and physical connectivity happen at the lowest level (i.e., port level).

To ensure interworking between data links with different multiplexing capabilities, LMP-capable devices SHOULD allow sub-channels of a component link to be locally configured as (logical) data links. For example, if a Router with 4 OC-48 interfaces is connected through a 4:1 MUX to a cross-connect with OC-192 interfaces, the cross-connect should be able to configure each sub-channel (e.g., STS-48c SPE if the 4:1 MUX is a SONET MUX) as a data link.

LMP is designed to support aggregation of one or more data links into a TE link (either ports into TE links, or component links into TE links). The purpose of forming a TE link is to group/map the information about certain physical resources (and their properties) into the information that is used by Constrained SPF for the purpose of path computation, and by GMPLS signaling.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The reader is assumed to be familiar with the terminology in [RFC3471], [RFC4202], and [RFC4201].

Bundled Link:

As defined in [RFC4201], a bundled link is a TE link such that, for the purpose of GMPLS signaling, a combination of <link identifier, label> is not sufficient to unambiguously identify the appropriate resources used by an LSP. A bundled link is composed of two or more component links.

Control Channel:

A control channel is a pair of mutually reachable interfaces that are used to enable communication between nodes for routing, signaling, and link management.

Component Link:

As defined in [RFC4201], a component link is a subset of resources of a TE Link such that (a) the partition is minimal, and (b) within each subset a label is sufficient to unambiguously identify the appropriate resources used by an LSP.

Data Link:

A data link is a pair of interfaces that are used to transfer user data. Note that in GMPLS, the control channel(s) between two adjacent nodes are no longer required to use the same physical medium as the data links between those nodes.

Link Property Correlation:

This is a procedure to correlate the local and remote properties of a TE link.

Multiplex Capability:

The ability to multiplex/demultiplex a data stream into sub-rate streams for switching purposes.

Node_Id:

For a node running OSPF, the LMP Node_Id is the same as the address contained in the OSPF Router Address TLV. For a node running IS-IS and advertising the TE Router ID TLV, the Node_Id is the same as the advertised Router ID.

Port:

An interface that terminates a data link.

TE Link:

As defined in [RFC4202], a TE link is a logical construct that represents a way to group/map the information about certain physical resources (and their properties) that interconnect LSRs into the information that is used by Constrained SPF for the purpose of path computation, and by GMPLS signaling.

Transparent:

A device is called X-transparent if it forwards incoming signals from input to output without examining or modifying the X aspect of the signal. For example, a Frame Relay switch is network-layer transparent; an all-optical switch is electrically transparent.

2. LMP Overview

The two core procedures of LMP are control channel management and link property correlation. Control channel management is used to establish and maintain control channels between adjacent nodes. This is done using a Config message exchange and a fast keep-alive mechanism between the nodes. The latter is required if lower-level mechanisms are not available to detect control channel failures. Link property correlation is used to synchronize the TE link properties and verify the TE link configuration.

LMP requires that a pair of nodes have at least one active bi-directional control channel between them. Each direction of the control channel is identified by a Control Channel Id (CC_Id), and the two directions are coupled together using the LMP Config message exchange. Except for Test messages, which may be limited by the

transport mechanism for in-band messaging, all LMP packets are run over UDP with an LMP port number. The link level encoding of the control channel is outside the scope of this document.

An "LMP adjacency" is formed between two nodes when at least one bi-directional control channel is established between them. Multiple control channels may be active simultaneously for each adjacency; control channel parameters, however, MUST be individually negotiated for each control channel. If the LMP fast keep-alive is used over a control channel, LMP Hello messages MUST be exchanged over the control channel. Other LMP messages MAY be transmitted over any of the active control channels between a pair of adjacent nodes. One or more active control channels may be grouped into a logical control channel for signaling, routing, and link property correlation purposes.

The link property correlation function of LMP is designed to aggregate multiple data links (ports or component links) into a TE link and to synchronize the properties of the TE link. As part of the link property correlation function, a LinkSummary message exchange is defined. The LinkSummary message includes the local and remote Link_Ids, a list of all data links that comprise the TE link, and various link properties. A LinkSummaryAck or LinkSummaryNack message MUST be sent in response to the receipt of a LinkSummary message indicating agreement or disagreement on the link properties.

LMP messages are transmitted reliably using Message_Ids and retransmissions. Message_Ids are carried in MESSAGE_ID objects. No more than one MESSAGE_ID object may be included in an LMP message. For control-channel-specific messages, the Message_Id is within the scope of the control channel over which the message is sent. For TE-link-specific messages, the Message_Id is within the scope of the LMP adjacency. The value of the Message_Id is monotonically increasing and wraps when the maximum value is reached.

In this document, two additional LMP procedures are defined: link connectivity verification and fault management. These procedures are particularly useful when the control channels are physically diverse from the data links. Link connectivity verification is used for data plane discovery, Interface_Id exchange (Interface_Ids are used in GMPLS signaling, either as port labels or component link identifiers, depending on the configuration), and physical connectivity verification. This is done by sending Test messages over the data links and TestStatus messages back over the control channel. Note that the Test message is the only LMP message that must be transmitted over the data link. The ChannelStatus message exchange is used between adjacent nodes for both the suppression of downstream alarms and the localization of faults for protection and restoration.

For LMP link connectivity verification, the Test message is transmitted over the data links. For X-transparent devices, this requires examining and modifying the X aspect of the signal. The LMP link connectivity verification procedure is coordinated using a BeginVerify message exchange over a control channel. To support various aspects of transparency, a Verify Transport Mechanism is included in the BeginVerify and BeginVerifyAck messages. Note that there is no requirement that all data links must lose their transparency simultaneously; but, at a minimum, it must be possible to terminate them one at a time. There is also no requirement that the control channel and TE link use the same physical medium; however, the control channel MUST be terminated by the same two control elements that control the TE link. Since the BeginVerify message exchange coordinates the Test procedure, it also naturally coordinates the transition of the data links in and out of the transparent mode.

The LMP fault management procedure is based on a ChannelStatus message exchange that uses the following messages: ChannelStatus, ChannelStatusAck, ChannelStatusRequest, and ChannelStatusResponse. The ChannelStatus message is sent unsolicited and is used to notify an LMP neighbor about the status of one or more data channels of a TE link. The ChannelStatusAck message is used to acknowledge receipt of the ChannelStatus message. The ChannelStatusRequest message is used to query an LMP neighbor for the status of one or more data channels of a TE Link. The ChannelStatusResponse message is used to acknowledge receipt of the ChannelStatusRequest message and indicate the states of the queried data links.

3. Control Channel Management

To initiate an LMP adjacency between two nodes, one or more bi-directional control channels MUST be activated. The control channels can be used to exchange control-plane information such as link provisioning and fault management information (implemented using a messaging protocol such as LMP, proposed in this document), path management and label distribution information (implemented using a signaling protocol such as RSVP-TE [RFC3209]), and network topology and state distribution information (implemented using traffic engineering extensions of protocols such as OSPF [RFC3630] and IS-IS [RFC3784]).

For the purposes of LMP, the exact implementation of the control channel is not specified; it could be, for example, a separate wavelength or fiber, an Ethernet link, an IP tunnel through a separate management network, or the overhead bytes of a data link. Each node assigns a node-wide, unique, 32-bit, non-zero integer control channel identifier (CC_Id). This identifier comes from the

same space as the unnumbered interface Id. Furthermore, LMP packets are run over UDP with an LMP port number. Thus, the link level encoding of the control channel is not part of the LMP specification.

To establish a control channel, the destination IP address on the far end of the control channel must be known. This knowledge may be manually configured or automatically discovered. Note that for in-band signaling, a control channel could be explicitly configured on a particular data link. In this case, the Config message exchange can be used to dynamically learn the IP address on the far end of the control channel. This is done by sending the Config message with the unicast IP source address and the multicast IP destination address (224.0.0.1 or ff02::1). The ConfigAck and ConfigNack messages MUST be sent to the source IP address found in the IP header of the received Config message.

Control channels exist independently of TE links and multiple control channels may be active simultaneously between a pair of nodes. Individual control channels can be realized in different ways; one might be implemented in-fiber while another one may be implemented out-of-fiber. As such, control channel parameters MUST be negotiated over each individual control channel, and LMP Hello packets MUST be exchanged over each control channel to maintain LMP connectivity if other mechanisms are not available. Since control channels are electrically terminated at each node, it may be possible to detect control channel failures using lower layers (e.g., SONET/SDH).

There are four LMP messages that are used to manage individual control channels. They are the Config, ConfigAck, ConfigNack, and Hello messages. These messages MUST be transmitted on the channel to which they refer. All other LMP messages may be transmitted over any of the active control channels between a pair of LMP adjacent nodes.

In order to maintain an LMP adjacency, it is necessary to have at least one active control channel between a pair of adjacent nodes (recall that multiple control channels can be active simultaneously between a pair of nodes). In the event of a control channel failure, alternate active control channels can be used and it may be possible to activate additional control channels as described below.

3.1. Parameter Negotiation

Control channel activation begins with a parameter negotiation exchange using Config, ConfigAck, and ConfigNack messages. The contents of these messages are built using LMP objects, which can be either negotiable or non-negotiable (identified by the N bit in the object header). Negotiable objects can be used to let LMP peers

agree on certain values. Non-negotiable objects are used for the announcement of specific values that do not need, or do not allow, negotiation.

To activate a control channel, a Config message MUST be transmitted to the remote node, and in response, a ConfigAck message MUST be received at the local node. The Config message contains the Local Control Channel Id (CC_Id), the sender's Node_Id, a Message_Id for reliable messaging, and a CONFIG object. It is possible that both the local and remote nodes initiate the configuration procedure at the same time. To avoid ambiguities, the node with the higher Node_Id wins the contention; the node with the lower Node_Id MUST stop transmitting the Config message and respond to the Config message it received. If the Node_Ids are equal, then one (or both) nodes have been misconfigured. The nodes MAY continue to retransmit Config messages in hopes that the misconfiguration is corrected. Note that the problem may be solved by an operator changing the Node_Ids on one or both nodes.

The ConfigAck message is used to acknowledge receipt of the Config message and express agreement on ALL of the configured parameters (both negotiable and non-negotiable).

The ConfigNack message is used to acknowledge receipt of the Config message, indicate which (if any) non-negotiable CONFIG objects are unacceptable, and to propose alternate values for the negotiable parameters.

If a node receives a ConfigNack message with acceptable alternate values for negotiable parameters, the node SHOULD transmit a Config message using these values for those parameters.

If a node receives a ConfigNack message with unacceptable alternate values, the node MAY continue to retransmit Config messages in hopes that the misconfiguration is corrected. Note that the problem may be solved by an operator changing parameters on one or both nodes.

In the case where multiple control channels use the same physical interface, the parameter negotiation exchange is performed for each control channel. The various LMP parameter negotiation messages are associated with their corresponding control channels by their node-wide unique identifiers (CC_Ids).

3.2. Hello Protocol

Once a control channel is activated between two adjacent nodes, the LMP Hello protocol can be used to maintain control channel connectivity between the nodes and to detect control channel

failures. The LMP Hello protocol is intended to be a lightweight keep-alive mechanism that will react to control channel failures rapidly so that IGP Hellos are not lost and the associated link-state adjacencies are not removed unnecessarily.

3.2.1. Hello Parameter Negotiation

Before sending Hello messages, the HelloInterval and HelloDeadInterval parameters MUST be agreed upon by the local and remote nodes. These parameters are exchanged in the Config message. The HelloInterval indicates how frequently LMP Hello messages will be sent, and is measured in milliseconds (ms). For example, if the value were 150, then the transmitting node would send the Hello message at least every 150 ms. The HelloDeadInterval indicates how long a device should wait to receive a Hello message before declaring a control channel dead, and is measured in milliseconds (ms).

The HelloDeadInterval MUST be greater than the HelloInterval, and SHOULD be at least 3 times the value of HelloInterval. If the fast keep-alive mechanism of LMP is not used, the HelloInterval and HelloDeadInterval parameters MUST be set to zero.

The values for the HelloInterval and HelloDeadInterval should be selected carefully to provide rapid response time to control channel failures without causing congestion. As such, different values will likely be configured for different control channel implementations. When the control channel is implemented over a directly connected link, the suggested default values for the HelloInterval is 150 ms and for the HelloDeadInterval is 500 ms.

When a node has either sent or received a ConfigAck message, it may begin sending Hello messages. Once it has sent a Hello message and received a valid Hello message (i.e., with expected sequence numbers; see Section 3.2.2), the control channel moves to the up state. (It is also possible to move to the up state without sending Hellos if other methods are used to indicate bi-directional control-channel connectivity. For example, indication of bi-directional connectivity may be learned from the transport layer.) If, however, a node receives a ConfigNack message instead of a ConfigAck message, the node MUST not send Hello messages and the control channel SHOULD NOT move to the up state. See Section 11.1 for the complete control channel FSM.

3.2.2. Fast Keep-alive

Each Hello message contains two sequence numbers: the first sequence number (TxSeqNum) is the sequence number for the Hello message being sent and the second sequence number (RcvSeqNum) is the sequence number of the last Hello message received from the adjacent node over this control channel.

There are two special sequence numbers. TxSeqNum MUST NOT ever be 0. TxSeqNum = 1 is used to indicate that the sender has just started or has restarted and has no recollection of the last TxSeqNum that was sent. Thus, the first Hello sent has a TxSeqNum of 1 and an RxSeqNum of 0. When TxSeqNum reaches (2^32)-1, the next sequence number used is 2, not 0 or 1, as these have special meanings.

Under normal operation, the difference between the RcvSeqNum in a Hello message that is received and the local TxSeqNum that is generated will be at most 1. This difference can be more than one only when a control channel restarts or when the values wrap.

Since the 32-bit sequence numbers may wrap, the following expression may be used to test if a newly received TxSeqNum value is less than a previously received value:

```
If ((int) old_id - (int) new_id > 0) {
   New value is less than old value;
}
```

Having sequence numbers in the Hello messages allows each node to verify that its peer is receiving its Hello messages. By including the RcvSeqNum in Hello packets, the local node will know which Hello packets the remote node has received.

The following example illustrates how the sequence numbers operate. Note that only the operation at one node is shown, and alternative scenarios are possible:

1) After completing the configuration stage, Node A sends Hello messages to Node B with {TxSeqNum=1;RcvSeqNum=0}.

2) Node A receives a Hello from Node B with {TxSeqNum=1;RcvSeqNum=1}. When the HelloInterval expires on Node A, it sends Hellos to Node B with {TxSeqNum=2;RcvSeqNum=1}.

3) Node A receives a Hello from Node B with {TxSeqNum=2;RcvSeqNum=2}. When the HelloInterval expires on Node A, it sends Hellos to Node B with {TxSeqNum=3;RcvSeqNum=2}.

3.2.3. Control Channel Down

To allow bringing a control channel down gracefully for administration purposes, a ControlChannelDown flag is available in the Common Header of LMP packets. When data links are still in use between a pair of nodes, a control channel SHOULD only be taken down administratively when there are other active control channels that can be used to manage the data links.

When bringing a control channel down administratively, a node MUST set the ControlChannelDown flag in all LMP messages sent over the control channel. The node that initiated the control channel down procedure may stop sending Hello messages after HelloDeadInterval seconds have passed, or if it receives an LMP message over the same control channel with the ControlChannelDown flag set.

When a node receives an LMP packet with the ControlChannelDown flag set, it SHOULD send a Hello message with the ControlChannelDown flag set and move the control channel to the down state.

3.2.4. Degraded State

A consequence of allowing the control channels to be physically diverse from the associated data links is that there may not be any active control channels available while the data links are still in use. For many applications, it is unacceptable to tear down a link that is carrying user traffic simply because the control channel is no longer available; however, the traffic that is using the data links may no longer be guaranteed the same level of service. Hence, the TE link is in a Degraded state.

When a TE link is in the Degraded state, routing and signaling SHOULD be notified so that new connections are not accepted and the TE link is advertised with no unreserved resources.

4. Link Property Correlation

As part of LMP, a link property correlation exchange is defined for TE links using the LinkSummary, LinkSummaryAck, and LinkSummaryNack messages. The contents of these messages are built using LMP objects, which can be either negotiable or non-negotiable (identified by the N flag in the object header). Negotiable objects can be used to let both sides agree on certain link parameters. Non-negotiable objects are used for announcement of specific values that do not need, or do not allow, negotiation.

Each TE link has an identifier (Link_Id) that is assigned at each end of the link. These identifiers MUST be the same type (i.e, IPv4, IPv6, unnumbered) at both ends. If a LinkSummary message is received with different local and remote TE link types, then a LinkSummaryNack message MUST be sent with Error Code "Bad TE Link Object". Similarly, each data link is assigned an identifier (Interface_Id) at each end. These identifiers MUST also be the same type at both ends. If a LinkSummary message is received with different local and remote Interface_Id types, then a LinkSummaryNack message MUST be sent with Error Code "Bad Data Link Object".

Link property correlation SHOULD be done before the link is brought up and MAY be done any time a link is up and not in the Verification process.

The LinkSummary message is used to verify for consistency the TE and data link information on both sides. Link Summary messages are also used (1) to aggregate multiple data links (either ports or component links) into a TE link; (2) to exchange, correlate (to determine inconsistencies), or change TE link parameters; and (3) to exchange, correlate (to determine inconsistencies), or change Interface_Ids (either Port_Ids or component link identifiers).

The LinkSummary message includes a TE_LINK object followed by one or more DATA_LINK objects. The TE_LINK object identifies the TE link's local and remote Link_Id and indicates support for fault management and link verification procedures for that TE link. The DATA_LINK objects are used to characterize the data links that comprise the TE link. These objects include the local and remote Interface_Ids, and may include one or more sub-objects further describing the properties of the data links.

If the LinkSummary message is received from a remote node, and the Interface_Id mappings match those that are stored locally, then the two nodes have agreement on the Verification procedure (see Section 5) and data link identification configuration. If the verification procedure is not used, the LinkSummary message can be used to verify agreement on manual configuration.

The LinkSummaryAck message is used to signal agreement on the Interface_Id mappings and link property definitions. Otherwise, a LinkSummaryNack message MUST be transmitted, indicating which Interface mappings are not correct and/or which link properties are not accepted. If a LinkSummaryNack message indicates that the Interface_Id mappings are not correct and the link verification procedure is enabled, the link verification process SHOULD be repeated for all mismatched, free data links; if an allocated data link has a mapping mismatch, it SHOULD be flagged and verified when

it becomes free. If a LinkSummaryNack message includes negotiable parameters, then acceptable values for those parameters MUST be included. If a LinkSummaryNack message is received and includes negotiable parameters, then the initiator of the LinkSummary message SHOULD send a new LinkSummary message. The new LinkSummary message SHOULD include new values for the negotiable parameters. These values SHOULD take into account the acceptable values received in the LinkSummaryNack message.

It is possible that the LinkSummary message could grow quite large due to the number of DATA LINK objects. An LMP implementation SHOULD be able to fragment when transmitting LMP messages, and MUST be able to re-assemble IP fragments when receiving LMP messages.

5. Verifying Link Connectivity

In this section, an optional procedure is described that may be used to verify the physical connectivity of the data links and dynamically learn (i.e., discover) the TE link and Interface_Id associations. The procedure SHOULD be done when establishing a TE link, and subsequently, on a periodic basis for all unallocated (free) data links of the TE link.

Support for this procedure is indicated by setting the "Link Verification Supported" flag in the TE_LINK object of the LinkSummary message.

If a BeginVerify message is received and link verification is not supported for the TE link, then a BeginVerifyNack message MUST be transmitted with Error Code indicating, "Link Verification Procedure not supported for this TE Link."

A unique characteristic of transparent devices is that the data is not modified or examined during normal operation. This characteristic poses a challenge for validating the connectivity of the data links and establishing the label mappings. Therefore, to ensure proper verification of data link connectivity, it is required that, until the data links are allocated for user traffic, they must be opaque (i.e., lose their transparency). To support various degrees of opaqueness (e.g., examining overhead bytes, terminating the IP payload, etc.) and, hence, different mechanisms to transport the Test messages, a Verify Transport Mechanism field is included in the BeginVerify and BeginVerifyAck messages.

There is no requirement that all data links be terminated simultaneously; but, at a minimum, the data links MUST be able to be terminated one at a time. Furthermore, for the link verification procedure it is assumed that the nodal architecture is designed so

that messages can be sent and received over any data link. Note that this requirement is trivial for opaque devices since each data link is electrically terminated and processed before being forwarded to the next opaque device; but that in transparent devices this is an additional requirement.

To interconnect two nodes, a TE link is defined between them, and at a minimum, there MUST be at least one active control channel between the nodes. For link verification, a TE link MUST include at least one data link.

Once a control channel has been established between the two nodes, data link connectivity can be verified by exchanging Test messages over each of the data links specified in the TE link. It should be noted that all LMP messages except the Test message are exchanged over the control channels and that Hello messages continue to be exchanged over each control channel during the data link verification process. The Test message is sent over the data link that is being verified. Data links are tested in the transmit direction because they are unidirectional; therefore, it may be possible for both nodes to (independently) exchange the Test messages simultaneously.

To initiate the link verification procedure, the local node MUST send a BeginVerify message over a control channel. To limit the scope of Link Verification to a particular TE Link, the local Link_Id MUST be non-zero. If this field is zero, the data links can span multiple TE links and/or they may comprise a TE link that is yet to be configured. For the case where the local Link_Id field is zero, the "Verify all Links" flag of the BEGIN_VERIFY object is used to distinguish between data links that span multiple TE links and those that have not yet been assigned to a TE link. Specifically, verification of data links that span multiple TE links is indicated by setting the local Link_Id field to zero and setting the "Verify all Links" flag. Verification of data links that have not yet been assigned to a TE link is indicated by setting the local Link_Id field to zero and clearing the "Verify all Links" flag.

The BeginVerify message also contains the number of data links that are to be verified; the interval (called VerifyInterval) at which the Test messages will be sent; the encoding scheme and transport mechanisms that are supported; the data rate for Test messages; and, when the data links correspond to fibers, the wavelength identifier over which the Test messages will be transmitted.

If the remote node receives a BeginVerify message and it is ready to process Test messages, it MUST send a BeginVerifyAck message back to the local node specifying the desired transport mechanism for the TEST messages. The remote node includes a 32-bit, node-unique

Verify_Id in the BeginVerifyAck message. The Verify_Id MAY be randomly selected; however, it MUST NOT overlap any other Verify_Id currently being used by the node selecting it. The Verify_Id is then used in all corresponding verification messages to differentiate them from different LMP peers and/or parallel Test procedures. When the local node receives a BeginVerifyAck message from the remote node, it may begin testing the data links by transmitting periodic Test messages over each data link. The Test message includes the Verify_Id and the local Interface_Id for the associated data link. The remote node MUST send either a TestStatusSuccess or a TestStatusFailure message in response for each data link. A TestStatusAck message MUST be sent to confirm receipt of the TestStatusSuccess and TestStatusFailure messages. Unacknowledged TestStatusSuccess and TestStatusFailure messages SHOULD be retransmitted until the message is acknowledged or until a retry limit is reached (see also Section 10).

It is also permissible for the sender to terminate the Test procedure anytime after sending the BeginVerify message. An EndVerify message SHOULD be sent for this purpose.

Message correlation is done using message identifiers and the Verify_Id; this enables verification of data links, belonging to different link bundles or LMP sessions, in parallel.

When the Test message is received, the received Interface_Id (used in GMPLS as either a Port label or component link identifier, depending on the configuration) is recorded and mapped to the local Interface_Id for that data link, and a TestStatusSuccess message MUST be sent. The TestStatusSuccess message includes the local Interface_Id along with the Interface_Id and Verify_Id received in the Test message. The receipt of a TestStatusSuccess message indicates that the Test message was detected at the remote node and the physical connectivity of the data link has been verified. When the TestStatusSuccess message is received, the local node SHOULD mark the data link as up and send a TestStatusAck message to the remote node. If, however, the Test message is not detected at the remote node within an observation period (specified by the VerifyDeadInterval), the remote node MUST send a TestStatusFailure message over the control channel, which indicates that the verification of the physical connectivity of the data link has failed. When the local node receives a TestStatusFailure message, it SHOULD mark the data link as FAILED and send a TestStatusAck message to the remote node. When all the data links on the list have been tested, the local node SHOULD send an EndVerify message to indicate that testing is complete on this link.

If the local/remote data link mappings are known, then the link verification procedure can be optimized by testing the data links in a defined order known to both nodes. The suggested criterion for this ordering is by increasing the value of the remote Interface_Id.

Both the local and remote nodes SHOULD maintain the complete list of Interface_Id mappings for correlation purposes.

5.1. Example of Link Connectivity Verification

Figure 1 shows an example of the link verification scenario that is executed when a link between Node A and Node B is added. In this example, the TE link consists of three free ports (each transmitted along a separate fiber) and is associated with a bi-directional control channel (indicated by a "c"). The verification process is as follows:

- o A sends a BeginVerify message over the control channel to B, indicating it will begin verifying the ports that form the TE link. The LOCAL_LINK_ID object carried in the BeginVerify message carries the identifier (IP address or interface index) that A assigns to the link.
- o Upon receipt of the BeginVerify message, B creates a Verify_Id and binds it to the TE Link from A. This binding is used later when B receives the Test messages from A, and these messages carry the Verify_Id. B discovers the identifier (IP address or interface index) that A assigns to the TE link by examining the LOCAL_LINK_ID object carried in the received BeginVerify message. (If the data ports are not yet assigned to the TE Link, the binding is limited to the Node_Id of A.) In response to the BeginVerify message, B sends the BeginVerifyAck message to A. The LOCAL_LINK_ID object carried in the BeginVerifyAck message is used to carry the identifier (IP address or interface index) that B assigns to the TE link. The REMOTE_LINK_ID object carried in the BeginVerifyAck message is used to bind the Link_Ids assigned by both A and B. The Verify_Id is returned to A in the BeginVerifyAck message over the control channel.
- o When A receives the BeginVerifyAck message, it begins transmitting periodic Test messages over the first port (Interface Id=1). The Test message includes the Interface_Id for the port and the Verify_Id that was assigned by B.
- o When B receives the Test messages, it maps the received Interface_Id to its own local Interface_Id = 10 and transmits a TestStatusSuccess message over the control channel back to Node A. The TestStatusSuccess message includes both the local and received Interface_Ids for the port as well as the Verify_Id. The

Verify_Id is used to determine the local/remote TE link identifiers (IP addresses or interface indices) to which the data links belong.

- o A will send a TestStatusAck message over the control channel back to B, indicating it received the TestStatusSuccess message.
- o The process is repeated until all of the ports are verified.
- o At this point, A will send an EndVerify message over the control channel to B, indicating that testing is complete.
- o B will respond by sending an EndVerifyAck message over the control channel back to A.

Note that this procedure can be used to "discover" the connectivity of the data ports.

```
   +---------------------+                      +---------------------+
   +                     +                      +                     +
   +      Node A         +<-------- c --------->+       Node B        +
   +                     +                      +                     +
   +                     +                      +                     +
   +                   1 +--------------------->+ 10                  +
   +                     +                      +                     +
   +                     +                      +                     +
   +                   2 +                /---->+ 11                  +
   +                     +          /----/      +                     +
   +                     +     /---/            +                     +
   +                   3 +----/                 + 12                  +
   +                     +                      +                     +
   +                     +                      +                     +
   +                   4 +--------------------->+ 14                  +
   +                     +                      +                     +
   +---------------------+                      +---------------------+
```

Figure 1: Example of link connectivity between Node A and Node B.

6. Fault Management

In this section, an optional LMP procedure is described that is used to manage failures by rapid notification of the status of one or more data channels of a TE Link. The scope of this procedure is within a TE link, and as such, the use of this procedure is negotiated as part of the LinkSummary exchange. The procedure can be used to rapidly isolate data link and TE link failures, and is designed to work for both unidirectional and bi-directional LSPs.

An important implication of using transparent devices is that traditional methods that are used to monitor the health of allocated data links may no longer be appropriate. Instead of fault detection being in layer 2 or layer 3, it is delegated to the physical layer (i.e., loss of light or optical monitoring of the data).

Recall that a TE link connecting two nodes may consist of a number of data links. If one or more data links fail between two nodes, a mechanism must be used for rapid failure notification so that appropriate protection/restoration mechanisms can be initiated. If the failure is subsequently cleared, then a mechanism must be used to notify that the failure is clear and the channel status is OK.

6.1. Fault Detection

Fault detection should be handled at the layer closest to the failure; for optical networks, this is the physical (optical) layer. One measure of fault detection at the physical layer is detecting loss of light (LOL). Other techniques for monitoring optical signals are still being developed and will not be further considered in this document. However, it should be clear that the mechanism used for fault notification in LMP is independent of the mechanism used to detect the failure, and simply relies on the fact that a failure is detected.

6.2. Fault Localization Procedure

In some situations, a data link failure between two nodes is propagated downstream such that all the downstream nodes detect the failure without localizing the failure. To avoid multiple alarms stemming from the same failure, LMP provides failure notification through the ChannelStatus message. This message may be used to indicate that a single data channel has failed, multiple data channels have failed, or an entire TE link has failed. Failure correlation is done locally at each node upon receipt of the failure notification.

To localize a fault to a particular link between adjacent nodes, a downstream node (downstream in terms of data flow) that detects data link failures will send a ChannelStatus message to its upstream neighbor indicating that a failure has been detected (bundling together the notification of all the failed data links). An upstream node that receives the ChannelStatus message MUST send a ChannelStatusAck message to the downstream node indicating it has received the ChannelStatus message. The upstream node should correlate the failure to see if the failure is also detected locally for the corresponding LSP(s). If, for example, the failure is clear on the input of the upstream node or internally, then the upstream

node will have localized the failure. Once the failure is correlated, the upstream node SHOULD send a ChannelStatus message to the downstream node indicating that the channel is failed or is OK. If a ChannelStatus message is not received by the downstream node, it SHOULD send a ChannelStatusRequest message for the channel in question. Once the failure has been localized, the signaling protocols may be used to initiate span or path protection and restoration procedures.

If all of the data links of a TE link have failed, then the upstream node MAY be notified of the TE link failure without specifying each data link of the failed TE link. This is done by sending failure notification in a ChannelStatus message identifying the TE Link without including the Interface_Ids in the CHANNEL_STATUS object.

6.3. Examples of Fault Localization

In Figure 2, a sample network is shown where four nodes are connected in a linear array configuration. The control channels are bi-directional and are labeled with a "c". All LSPs are also bi-directional.

In the first example [see Fig. 2(a)], there is a failure on one direction of the bi-directional LSP. Node 4 will detect the failure and will send a ChannelStatus message to Node 3 indicating the failure (e.g., LOL) to the corresponding upstream node. When Node 3 receives the ChannelStatus message from Node 4, it returns a ChannelStatusAck message back to Node 4 and correlates the failure locally. When Node 3 correlates the failure and verifies that the failure is clear, it has localized the failure to the data link between Node 3 and Node 4. At that time, Node 3 should send a ChannelStatus message to Node 4 indicating that the failure has been localized.

In the second example [see Fig. 2(b)], a single failure (e.g., fiber cut) affects both directions of the bi-directional LSP. Node 2 (Node 3) will detect the failure of the upstream (downstream) direction and send a ChannelStatus message to the upstream (in terms of data flow) node indicating the failure (e.g., LOL). Simultaneously (ignoring propagation delays), Node 1 (Node 4) will detect the failure on the upstream (downstream) direction, and will send a ChannelStatus message to the corresponding upstream (in terms of data flow) node indicating the failure. Node 2 and Node 3 will have localized the two directions of the failure.

```
       +-------+        +-------+        +-------+        +-------+
       + Node1 +        + Node2 +        + Node3 +        + Node4 +
       +       +-- c ---+       +-- c ---+       +-- c ---+       +
   ----+---\   +        +       +        +       +        +       +
   <---+---\\--+--------+-------+---\    +       +        +    /--+--->
       +    \--+--------+-------+---\\---+-------+---##---+---//--+----
       +       +        +       +    \---+-------+--------+---/   +
       +       +        +       +        +       +  (a)   +       +
   ----+-------+--------+---\   +        +       +        +       +
   <---+-------+--------+---\\--+---##---+--\    +        +       +
       +       +        +    \--+---##---+--\\   +        +       +
       +       +        +       +  (b)   +   \\--+--------+-------+--->
       +       +        +       +        +    \--+--------+-------+----
       +       +        +       +        +       +        +       +
       +-------+        +-------+        +-------+        +-------+
```

Figure 2: Two types of data link failures are shown (indicated by ## in the figure):
(A) a data link corresponding to the downstream direction of a bi-directional LSP fails,
(B) two data links corresponding to both directions of a bi-directional LSP fail. The control channel connecting two nodes is indicated with a "c".

6.4. Channel Activation Indication

The ChannelStatus message may also be used to notify an LMP neighbor that the data link should be actively monitored. This is called Channel Activation Indication. This is particularly useful in networks with transparent nodes where the status of data links may need to be triggered using control channel messages. For example, if a data link is pre-provisioned and the physical link fails after verification and before inserting user traffic, a mechanism is needed to indicate the data link should be active, otherwise the failure may not be detectable.

The ChannelStatus message is used to indicate that a channel or group of channels are now active. The ChannelStatusAck message MUST be transmitted upon receipt of a ChannelStatus message. When a ChannelStatus message is received, the corresponding data link(s) MUST be put into the Active state. If upon putting them into the Active state, a failure is detected, the ChannelStatus message SHOULD be transmitted as described in Section 6.2.

6.5. Channel Deactivation Indication

The ChannelStatus message may also be used to notify an LMP neighbor that the data link no longer needs to be actively monitored. This is the counterpart to the Channel Active Indication.

When a ChannelStatus message is received with Channel Deactive Indication, the corresponding data link(s) MUST be taken out of the Active state.

7. Message_Id Usage

The MESSAGE_ID and MESSAGE_ID_ACK objects are included in LMP messages to support reliable message delivery. This section describes the usage of these objects. The MESSAGE_ID and MESSAGE_ID_ACK objects contain a Message_Id field.

Only one MESSAGE_ID/MESSAGE_ID_ACK object may be included in any LMP message.

For control-channel-specific messages, the Message_Id field is within the scope of the CC_Id. For TE link specific messages, the Message_Id field is within the scope of the LMP adjacency.

The Message_Id field of the MESSAGE_ID object contains a generator-selected value. This value MUST be monotonically increasing. A value is considered to be previously used when it has been sent in an LMP message with the same CC_Id (for control channel specific messages) or LMP adjacency (for TE Link specific messages). The Message_Id field of the MESSAGE_ID_ACK object contains the Message_Id field of the message being acknowledged.

Unacknowledged messages sent with the MESSAGE_ID object SHOULD be retransmitted until the message is acknowledged or until a retry limit is reached (see also Section 10).

Note that the 32-bit Message_Id value may wrap. The following expression may be used to test if a newly received Message_Id value is less than a previously received value:

```
If ((int) old_id - (int) new_id > 0) {
   New value is less than old value;
}
```

Nodes processing incoming messages SHOULD check to see if a newly received message is out of order and can be ignored. Out-of-order messages can be identified by examining the value in the Message_Id field. If a message is determined to be out-of-order, that message should be silently dropped.

If the message is a Config message, and the Message_Id value is less than the largest Message_Id value previously received from the sender for the CC_Id, then the message SHOULD be treated as being out-of-order.

If the message is a LinkSummary message and the Message_Id value is less than the largest Message_Id value previously received from the sender for the TE Link, then the message SHOULD be treated as being out-of-order.

If the message is a ChannelStatus message and the Message_Id value is less than the largest Message_Id value previously received from the sender for the specified TE link, then the receiver SHOULD check the Message_Id value previously received for the state of each data channel included in the ChannelStatus message. If the Message_Id value is greater than the most recently received Message_Id value associated with at least one of the data channels included in the message, the message MUST NOT be treated as out of order; otherwise, the message SHOULD be treated as being out of order. However, the state of any data channel MUST NOT be updated if the Message_Id value is less than the most recently received Message_Id value associated with the data channel.

All other messages MUST NOT be treated as out-of-order.

8. Graceful Restart

This section describes the mechanism to resynchronize the LMP state after a control plane restart. A control plane restart may occur when bringing up the first control channel after a control communications failure. A control communications failure may be the result of an LMP adjacency failure or a nodal failure wherein the LMP control state is lost, but the data plane is unaffected. The latter is detected by setting the "LMP Restart" bit in the Common Header of the LMP messages. When the control plane fails due to the loss of the control channel, the LMP link information should be retained. It is possible that a node may be capable of retaining the LMP link information across a nodal failure. However, in both cases the status of the data channels MUST be synchronized.

It is assumed the Node_Id and Local Interface_Ids remain stable across a control plane restart.

After the control plane of a node restarts, the control channel(s) must be re-established using the procedures of Section 3.1. When re-establishing control channels, the Config message SHOULD be sent using the unicast IP source and destination addresses.

If the control plane failure was the result of a nodal failure where the LMP control state is lost, then the "LMP Restart" flag MUST be set in LMP messages until a Hello message is received with the RcvSeqNum equal to the local TxSeqNum. This indicates that the control channel is up and the LMP neighbor has detected the restart.

The following assumes that the LMP component restart only occurred on one end of the TE Link. If the LMP component restart occurred on both ends of the TE Link, the normal procedures for LinkSummary should be used, as described in Section 4.

Once a control channel is up, the LMP neighbor MUST send a LinkSummary message for each TE Link across the adjacency. All the objects of the LinkSummary message MUST have the N-bit set to 0, indicating that the parameters are non-negotiable. This provides the local/remote Link_Id and Interface_Id mappings, the associated data link parameters, and indication of which data links are currently allocated to user traffic. When a node receives the LinkSummary message, it checks its local configuration. If the node is capable of retaining the LMP link information across a restart, it must process the LinkSummary message as described in Section 4 with the exception that the allocated/de-allocated flag of the DATA_LINK object received in the LinkSummary message MUST take precedence over any local value. If, however, the node was not capable of retaining the LMP link information across a restart, the node MUST accept the data link parameters of the received LinkSummary message and respond with a LinkSummaryAck message.

Upon completion of the LinkSummary exchange, the node that has restarted the control plane SHOULD send a ChannelStatusRequest message for that TE link. The node SHOULD also verify the connectivity of all unallocated data channels.

9. Addressing

All LMP messages are run over UDP with an LMP port number (except, in some cases, the Test messages, which may be limited by the transport mechanism for in-band messaging). The destination address of the IP packet MAY be either the address learned in the Configuration procedure (i.e., the Source IP address found in the IP header of the

received Config message), an IP address configured on the remote node, or the Node_Id. The Config message is an exception as described below.

The manner in which a Config message is addressed may depend on the signaling transport mechanism. When the transport mechanism is a point-to-point link, Config messages SHOULD be sent to the Multicast address (224.0.0.1 or ff02::1). Otherwise, Config messages MUST be sent to an IP address on the neighboring node. This may be configured at both ends of the control channel or may be automatically discovered.

10. Exponential Back-off Procedures

This section is based on [RFC2961] and provides exponential back-off procedures for message retransmission. Implementations MUST use the described procedures or their equivalent.

10.1. Operation

The following operation is one possible mechanism for exponential back-off retransmission of unacknowledged LMP messages. The sending node retransmits the message until an acknowledgement message is received or until a retry limit is reached. When the sending node receives the acknowledgement, retransmission of the message is stopped. The interval between message retransmission is governed by a rapid retransmission timer. The rapid retransmission timer starts at a small interval and increases exponentially until it reaches a threshold.

The following time parameters are useful to characterize the procedures:

Rapid retransmission interval Ri:

Ri is the initial retransmission interval for unacknowledged messages. After sending the message for the first time, the sending node will schedule a retransmission after Ri milliseconds.

Rapid retry limit Rl:

Rl is the maximum number of times a message will be transmitted without being acknowledged.

Increment value Delta:

Delta governs the speed with which the sender increases the retransmission interval. The ratio of two successive retransmission intervals is (1 + Delta).

Suggested default values for an initial retransmission interval (Ri) of 500 ms are a power of 2 exponential back-off (Delta = 1) and a retry limit of 3.

10.2. Retransmission Algorithm

After a node transmits a message requiring acknowledgement, it should immediately schedule a retransmission after Ri seconds. If a corresponding acknowledgement message is received before Ri seconds, then message retransmission SHOULD be canceled. Otherwise, it will retransmit the message after (1+Delta)*Ri seconds. The retransmission will continue until either an appropriate acknowledgement message is received or the rapid retry limit, Rl, has been reached.

A sending node can use the following algorithm when transmitting a message that requires acknowledgement:

```
Prior to initial transmission, initialize Rk = Ri and Rn = 0.

while (Rn++ < Rl) {
  transmit the message;
  wake up after Rk milliseconds;
  Rk = Rk * (1 + Delta);
}
/* acknowledged message or no reply from receiver and Rl
reached*/
do any needed clean up;
exit;
```

Asynchronously, when a sending node receives a corresponding acknowledgment message, it will change the retry count, Rn, to Rl.

Note that the transmitting node does not advertise or negotiate the use of the described exponential back-off procedures in the Config or LinkSummary messages.

11. LMP Finite State Machines

11.1. Control Channel FSM

The control channel FSM defines the states and logics of operation of an LMP control channel.

11.1.1. Control Channel States

A control channel can be in one of the states described below. Every state corresponds to a certain condition of the control channel and is usually associated with a specific type of LMP message that is periodically transmitted to the far end.

Down: This is the initial control channel state. In this state, no attempt is being made to bring the control channel up and no LMP messages are sent. The control channel parameters should be set to the initial values.

ConfSnd: The control channel is in the parameter negotiation state. In this state the node periodically sends a Config message, and is expecting the other side to reply with either a ConfigAck or ConfigNack message. The FSM does not transition into the Active state until the remote side positively acknowledges the parameters.

ConfRcv: The control channel is in the parameter negotiation state. In this state, the node is waiting for acceptable configuration parameters from the remote side. Once such parameters are received and acknowledged, the FSM can transition to the Active state.

Active: In this state the node periodically sends a Hello message and is waiting to receive a valid Hello message. Once a valid Hello message is received, it can transition to the up state.

Up: The CC is in an operational state. The node receives valid Hello messages and sends Hello messages.

GoingDown: A CC may go into this state because of administrative action. While a CC is in this state, the node sets the ControlChannelDown bit in all the messages it sends.

11.1.2. Control Channel Events

Operation of the LMP control channel is described in terms of FSM states and events. Control channel events are generated by the underlying protocols and software modules, as well as by the packet processing routines and FSMs of associated TE links. Every event has its number and a symbolic name. Description of possible control channel events is given below.

```
1 : evBringUp:    This is an externally triggered event indicating
                  that the control channel negotiation should begin.
                  This event, for example, may be triggered by an
                  operator command, by the successful completion of a
                  control channel bootstrap procedure, or by
                  configuration.  Depending on the configuration,
                  this will trigger either
                      1a)  the sending of a Config message,
                      1b)  a period of waiting to receive a Config
                           message from the remote node.

2 : evCCDn:       This event is generated when there is indication
                  that the control channel is no longer available.

3 : evConfDone:   This event indicates a ConfigAck message has been
                  received, acknowledging the Config parameters.

4 : evConfErr:    This event indicates a ConfigNack message has been
                  received, rejecting the Config parameters.

5 : evNewConfOK:  New Config message was received from neighbor and
                  positively acknowledged.

6 : evNewConfErr: New Config message was received from neighbor and
                  rejected with a ConfigNack message.

7 : evContenWin:  New Config message was received from neighbor at
                  the same time a Config message was sent to the
                  neighbor.  The local node wins the contention.  As
                  a result, the received Config message is ignored.

8 : evContenLost: New Config message was received from neighbor at
                  the same time a Config message was sent to the
                  neighbor.  The local node loses the contention.
                      8a)  The Config message is positively
                           acknowledged.
                      8b)  The Config message is negatively
                           acknowledged.
```

9 : evAdminDown: The administrator has requested that the control channel is brought down administratively.

10: evNbrGoesDn: A packet with ControlChannelDown flag is received from the neighbor.

11: evHelloRcvd: A Hello packet with expected SeqNum has been received.

12: evHoldTimer: The HelloDeadInterval timer has expired indicating that no Hello packet has been received. This moves the control channel back into the Negotiation state, and depending on the local configuration, this will trigger either
  12a) the sending of periodic Config messages,
  12b) a period of waiting to receive Config messages from the remote node.

13: evSeqNumErr: A Hello with unexpected SeqNum received and discarded.

14: evReconfig: Control channel parameters have been reconfigured and require renegotiation.

15: evConfRet: A retransmission timer has expired and a Config message is resent.

16: evHelloRet: The HelloInterval timer has expired and a Hello packet is sent.

17: evDownTimer: A timer has expired and no messages have been received with the ControlChannelDown flag set.

11.1.3. Control Channel FSM Description

Figure 3 illustrates operation of the control channel FSM in a form of FSM state transition diagram.

```
                               +--------+
            +----------------->|        |<--------------+
            |       +--------->|  Down  |<----------+   |
            |       ||+---------|        |<-------+  |   |
            |       ||         +--------+        |  |   |
            |       ||           |    ^       2,9| 2|  2|
            |       ||1b       1a|    |          |  |   |
            |       ||           v    |2,9       |  |   |
            |       ||         +--------+        |  |   |
            |       ||      +->|        |<------+|  |   |
            |       ||  4,7,|  |ConfSnd |       ||  |   |
            |       || 14,15+--|        |<----+ ||  |   |
            |       ||         +--------+     | ||  |   |
            |       ||       3,8a| |          | ||  |   |
            |       || +---------+ |8b  14,12a| ||  |   |
            |       || |           v          | ||  |   |
            |       |+-|------>+--------+     | ||  |   |
            |       |  |    +->|        |-----|-|+  |   |
            |       |  |6,14|  |ConfRcv |     | |   |   |
            |       |  |    +--|        |<--+ | |   |   |
            |       |  |       +--------+   | | |   |   |
            |       |  |          5| ^      | | |   |   |
            |       |  +---------+ | |      | | |   |   |
            |       |            | | |      | | |   |   |
            |       |            v v |6,12b | | |   |   |
            |       |10        +--------+   | | |   |   |
            |       +----------|        |   | | |   |   |
            |       |       +--| Active |---|-+ |   |   |
       10,17|       |   5,16|  |        |-------|---+   |
        +-------+ 9 |   13  +->|        |   |   |       |
        | Going |<--|----------+--------+   |   |       |
        | Down  |   |           11| ^       |   |       |
        +-------+   |             | |5      |   |       |
            ^       |             v |  6,12b|   |       |
            |9      |10        +--------+   |   |12a,14 |
            |       +----------|        |---+   |       |
            |                  |   Up   |-------+       |
            +------------------|        |---------------+
                               +--------+
                                 |   ^
                                 |   |
                                 +---+
                                11,13,16
```

Figure 3: Control Channel FSM

Event evCCDn always forces the FSM to the down state. Events evHoldTimer and evReconfig always force the FSM to the Negotiation state (either ConfSnd or ConfRcv).

11.2. TE Link FSM

The TE Link FSM defines the states and logics of operation of the LMP TE Link.

11.2.1. TE Link States

An LMP TE link can be in one of the states described below. Every state corresponds to a certain condition of the TE link and is usually associated with a specific type of LMP message that is periodically transmitted to the far end via the associated control channel or in-band via the data links.

Down: There are no data links allocated to the TE link.

Init: Data links have been allocated to the TE link, but the configuration has not yet been synchronized with the LMP neighbor. The LinkSummary message is periodically transmitted to the LMP neighbor.

Up: This is the normal operational state of the TE link. At least one LMP control channel is required to be operational between the nodes sharing the TE link. As part of normal operation, the LinkSummary message may be periodically transmitted to the LMP neighbor or generated by an external request.

Degraded: In this state, all LMP control channels are down, but the TE link still includes some data links that are allocated to user traffic.

11.2.2. TE Link Events

Operation of the LMP TE link is described in terms of FSM states and events. TE Link events are generated by the packet processing routines and by the FSMs of the associated control channel(s) and the data links. Every event has its number and a symbolic name. Descriptions of possible events are given below.

1 : evDCUp: One or more data channels have been enabled and assigned to the TE Link.

2 : evSumAck: LinkSummary message received and positively acknowledged.

3 : evSumNack: LinkSummary message received and negatively acknowledged.

4 : evRcvAck: LinkSummaryAck message received acknowledging the TE Link Configuration.

5 : evRcvNack: LinkSummaryNack message received.

6 : evSumRet: Retransmission timer has expired and LinkSummary message is resent.

7 : evCCUp: First active control channel goes up.

8 : evCCDown: Last active control channel goes down.

9 : evDCDown: Last data channel of TE Link has been removed.

11.2.3. TE Link FSM Description

Figure 4 illustrates operation of the LMP TE Link FSM in a form of FSM state transition diagram.

```
                                  3,7,8
                                   +--+
                                   |  |
                                   |  v
                                +--------+
                                |        |
                  +------------>|  Down  |<---------+
                  |             |        |          |
                  |             +--------+          |
                  |                |  ^             |
                  |               1|  |9            |
                  |                v  |             |
                  |             +--------+          |
                  |             |        |<-+       |
                  |             |  Init  |  |3,5,6  |9
                  |             |        |--+ 7,8   |
                 9|             +--------+          |
                  |                  |              |
                  |               2,4|              |
                  |                  v              |
               +--------+   7   +--------+          |
               |        |------>|        |----------+
               |  Deg   |       |   Up   |
               |        |<------|        |
               +--------+   8   +--------+
                                   |  ^
                                   |  |
                                   +--+
                                 2,3,4,5,6
```

Figure 4: LMP TE Link FSM

In the above FSM, the sub-states that may be implemented when the link verification procedure is used have been omitted.

11.3. Data Link FSM

The data link FSM defines the states and logics of operation of a data link within an LMP TE link. Operation of a data link is described in terms of FSM states and events. Data links can either be in the active (transmitting) mode, where Test messages are transmitted from them, or the passive (receiving) mode, where Test messages are received through them. For clarity, separate FSMs are defined for the active/passive data links; however, a single set of data link states and events are defined.

11.3.1. Data Link States

Any data link can be in one of the states described below. Every state corresponds to a certain condition of the data link.

Down: The data link has not been put in the resource pool (i.e., the link is not 'in service')

Test: The data link is being tested. An LMP Test message is periodically sent through the link.

PasvTest: The data link is being checked for incoming test messages.

Up/Free: The link has been successfully tested and is now put in the pool of resources (in-service). The link has not yet been allocated to data traffic.

Up/Alloc: The link is up and has been allocated for data traffic.

11.3.2. Data Link Events

Data link events are generated by the packet processing routines and by the FSMs of the associated control channel and the TE link.

Every event has its number and a symbolic name. Description of possible data link events is given below:

1 :evCCUp: First active control channel goes up.

2 :evCCDown: LMP neighbor connectivity is lost. This indicates the last LMP control channel has failed between neighboring nodes.

3 :evStartTst: This is an external event that triggers the sending of Test messages over the data link.

4 :evStartPsv: This is an external event that triggers the listening for Test messages over the data link.

5 :evTestOK: Link verification was successful and the link can be used for path establishment.
- (a) This event indicates the Link Verification procedure (see Section 5) was successful for this data link and a TestStatusSuccess message was received over the control channel.

```
                             (b)  This event indicates the link is ready for
                                  path establishment, but the Link
                                  Verification procedure was not used.  For
                                  in-band signaling of the control channel,
                                  the control channel establishment may be
                                  sufficient to verify the link.

   6 :evTestRcv:      Test message was received over the data port and a
                      TestStatusSuccess message is transmitted over the
                      control channel.

   7 :evTestFail:     Link verification returned negative results.  This
                      could be because (a) a TestStatusFailure message
                      was received, or (b) the Verification procedure
                      has ended without receiving a TestStatusSuccess or
                      TestStatusFailure message for the data link.

   8 :evPsvTestFail:  Link verification returned negative results.  This
                      indicates that a Test message was not detected and
                      either (a) the VerifyDeadInterval has expired or
                      (b) the Verification procedure has ended and the
                      VerifyDeadInterval has not yet expired.

   9 :evLnkAlloc:     The data link has been allocated.

   10:evLnkDealloc:   The data link has been de-allocated.

   11:evTestRet:      A retransmission timer has expired and the Test
                      message is resent.

   12:evSummaryFail:  The LinkSummary did not match for this data port.

   13:evLocalizeFail: A Failure has been localized to this data link.

   14:evdcDown:      The data channel is no longer available.
```

11.3.3. Active Data Link FSM Description

Figure 5 illustrates operation of the LMP active data link FSM in a form of FSM state transition diagram.

```
                             +------+
                             |      |<-------+
                  +--------->| Down |        |
                  |     +----|      |<-----+ |
                  |     |    +------+      | |
                  |     |5b   3|  ^        | |
                  |     |      |  |7       | |
                  |     |      v  |        | |
                  |     |    +------+      | |
                  |     |    |      |<-+   | |
                  |     |    | Test |  |11 | |
                  |     |    |      |--+   | |
                  |     |    +------+      | |
                  |     |     5a| 3^       | |
                  |     |       |  |       | |
                  |     |       v  |       | |
                  |12   |   +---------+    | |
                  |     +-->|         |14  | |
                  |         | Up/Free |----+ |
                  +---------|         |      |
                            +---------+      |
                               9| ^          |
                                | |          |
                                v |10        |
                            +---------+      |
                            |         |13    |
                            |Up/Alloc |------+
                            |         |
                            +---------+
```

Figure 5: Active LMP Data Link FSM

11.3.4. Passive Data Link FSM Description

Figure 6 illustrates operation of the LMP passive data link FSM in a form of FSM state transition diagram.

```
                          +------+
                          |      |<------+
              +---------->| Down |       |
              |     +-----|      |<----+ |
              |     |     +------+     | |
              |     |5b    4|  ^       | |
              |     |       |  |8      | |
              |     |       v  |       | |
              |     |    +----------+  | |
              |     |    | PasvTest |  | |
              |     |    +----------+  | |
              |     |       6|  4^     | |
              |     |        |   |     | |
              |     |        v   |     | |
              |12   |    +---------+   | |
              |     +--->| Up/Free |14 | |
              |          |         |---+ |
              +----------|         |     |
                         +---------+     |
                             9| ^        |
                              | |        |
                              v |10      |
                         +---------+     |
                         |         |13   |
                         |Up/Alloc |-----+
                         |         |
                         +---------+
```

Figure 6: Passive LMP Data Link FSM

12. LMP Message Formats

All LMP messages (except, in some cases, the Test messages, which are limited by the transport mechanism for in-band messaging) are run over UDP with an LMP port number (701).

12.1. Common Header

In addition to the UDP header and standard IP header, all LMP messages (except, in some cases, the Test messages which may be limited by the transport mechanism for in-band messaging) have the following common header:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vers  |      (Reserved)       |    Flags      |    Msg Type   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          LMP Length           |          (Reserved)           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Reserved field should be sent as zero and ignored on receipt.

All values are defined in network byte order (i.e., big-endian byte order).

Vers: 4 bits

Protocol version number. This is version 1.

Flags: 8 bits

The following bit-values are defined. All other bits are reserved and should be sent as zero and ignored on receipt.

0x01: ControlChannelDown

0x02: LMP Restart

This bit is set to indicate that a nodal failure has occurred and the LMP control state has been lost. This flag may be reset to 0 when a Hello message is received with RcvSeqNum equal to the local TxSeqNum.

Msg Type: 8 bits

The following values are defined. All other values are reserved

1 = Config

2 = ConfigAck

3 = ConfigNack

4 = Hello

5 = BeginVerify

6 = BeginVerifyAck

7 = BeginVerifyNack

8 = EndVerify

9 = EndVerifyAck

10 = Test

11 = TestStatusSuccess

12 = TestStatusFailure

13 = TestStatusAck

14 = LinkSummary

15 = LinkSummaryAck

16 = LinkSummaryNack

17 = ChannelStatus

18 = ChannelStatusAck

19 = ChannelStatusRequest

20 = ChannelStatusResponse

All of the messages are sent over the control channel EXCEPT the Test message, which is sent over the data link that is being tested.

LMP Length: 16 bits

The total length of this LMP message in bytes, including the common header and any variable-length objects that follow.

12.2. LMP Object Format

LMP messages are built using objects. Each object is identified by its Object Class and Class-type. Each object has a name, which is always capitalized in this document. LMP objects can be either negotiable or non-negotiable (identified by the N bit in the object header). Negotiable objects can be used to let the devices agree on certain values. Non-negotiable objects are used for announcement of specific values that do not need or do not allow negotiation.

All values are defined in network byte order (i.e., big-endian byte order).

The format of the LMP object is as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|N|   C-Type    |     Class     |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
//                       (object contents)                     //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

N: 1 bit

The N flag indicates if the object is negotiable (N=1) or non-negotiable (N=0).

C-Type: 7 bits

Class-type, unique within an Object Class. Values are defined in Section 13.

Class: 8 bits

The Class indicates the object type. Each object has a name, which is always capitalized in this document.

Length: 16 bits

The Length field indicates the length of the object in bytes, including the N, C-Type, Class, and Length fields.

12.3. Parameter Negotiation Messages

12.3.1. Config Message (Msg Type = 1)

The Config message is used in the control channel negotiation phase of LMP. The contents of the Config message are built using LMP objects. The format of the Config message is as follows:

```
<Config Message> ::= <Common Header> <LOCAL_CCID> <MESSAGE_ID>
                     <LOCAL_NODE_ID> <CONFIG>
```

The above transmission order SHOULD be followed.

The MESSAGE_ID object is within the scope of the LOCAL_CCID object.

The Config message MUST be periodically transmitted until (1) it receives a ConfigAck or ConfigNack message, (2) a retry limit has been reached and no ConfigAck or ConfigNack message has been received, or (3) it receives a Config message from the remote node and has lost the contention (e.g., the Node_Id of the remote node is higher than the Node_Id of the local node). Both the retransmission interval and the retry limit are local configuration parameters.

12.3.2. ConfigAck Message (Msg Type = 2)

The ConfigAck message is used to acknowledge receipt of the Config message and indicate agreement on all parameters.

```
<ConfigAck Message> ::= <Common Header> <LOCAL_CCID> <LOCAL_NODE_ID>
                        <REMOTE_CCID> <MESSAGE_ID_ACK>
                        <REMOTE_NODE_ID>
```

The above transmission order SHOULD be followed.

The contents of the REMOTE_CCID, MESSAGE_ID_ACK, and REMOTE_NODE_ID objects MUST be obtained from the Config message being acknowledged.

12.3.3. ConfigNack Message (Msg Type = 3)

The ConfigNack message is used to acknowledge receipt of the Config message and indicate disagreement on non-negotiable parameters or propose other values for negotiable parameters. Parameters where agreement was reached MUST NOT be included in the ConfigNack Message. The format of the ConfigNack message is as follows:

```
<ConfigNack Message> ::= <Common Header> <LOCAL_CCID>
                         <LOCAL_NODE_ID>  <REMOTE_CCID>
                         <MESSAGE_ID_ACK> <REMOTE_NODE_ID> <CONFIG>
```

The above transmission order SHOULD be followed.

The contents of the REMOTE_CCID, MESSAGE_ID_ACK, and REMOTE_NODE_ID objects MUST be obtained from the Config message being negatively acknowledged.

It is possible that multiple parameters may be invalid in the Config message.

If a negotiable CONFIG object is included in the ConfigNack message, it MUST include acceptable values for the parameters.

If the ConfigNack message includes CONFIG objects for non-negotiable parameters, they MUST be copied from the CONFIG objects received in the Config message.

If the ConfigNack message is received and only includes CONFIG objects that are negotiable, then a new Config message SHOULD be sent. The values in the CONFIG object of the new Config message SHOULD take into account the acceptable values included in the ConfigNack message.

If a node receives a Config message and recognizes the CONFIG object, but does not recognize the C-Type, a ConfigNack message including the unknown CONFIG object MUST be sent.

12.4. Hello Message (Msg Type = 4)

The format of the Hello message is as follows:

```
<Hello Message> ::= <Common Header> <LOCAL_CCID> <HELLO>
```

The above transmission order SHOULD be followed.

The Hello message MUST be periodically transmitted at least once every HelloInterval msec. If no Hello message is received within the HelloDeadInterval, the control channel is assumed to have failed.

12.5. Link Verification Messages

12.5.1. BeginVerify Message (Msg Type = 5)

The BeginVerify message is sent over the control channel and is used to initiate the link verification process. The format is as follows:

```
<BeginVerify Message> ::= <Common Header> <LOCAL_LINK_ID>
                          <MESSAGE_ID> [<REMOTE_LINK_ID>]
                          <BEGIN_VERIFY>
```

The above transmission order SHOULD be followed.

To limit the scope of Link Verification to a particular TE Link, the Link_Id field of the LOCAL_LINK_ID object MUST be non-zero. If this field is zero, the data links can span multiple TE links and/or they may comprise a TE link that is yet to be configured. In the special case where the local Link_Id field is zero, the "Verify all Links" flag of the BEGIN_VERIFY object is used to distinguish between data links that span multiple TE links and those that have not yet been assigned to a TE link (see Section 5).

The REMOTE_LINK_ID object may be included if the local/remote Link_Id mapping is known.

The Link_Id field of the REMOTE_LINK_ID object MUST be non-zero if included.

The BeginVerify message MUST be periodically transmitted until (1) the node receives either a BeginVerifyAck or BeginVerifyNack message to accept or reject the verify process or (2) a retry limit has been reached and no BeginVerifyAck or BeginVerifyNack message has been received. Both the retransmission interval and the retry limit are local configuration parameters.

12.5.2. BeginVerifyAck Message (Msg Type = 6)

When a BeginVerify message is received and Test messages are ready to be processed, a BeginVerifyAck message MUST be transmitted.

```
<BeginVerifyAck Message> ::= <Common Header> [<LOCAL_LINK_ID>]
                             <MESSAGE_ID_ACK> <BEGIN_VERIFY_ACK>
                             <VERIFY_ID>
```

The above transmission order SHOULD be followed.

The LOCAL_LINK_ID object may be included if the local/remote Link_Id mapping is known or learned through the BeginVerify message.

The Link_Id field of the LOCAL_LINK_ID MUST be non-zero if included.

The contents of the MESSAGE_ID_ACK object MUST be obtained from the BeginVerify message being acknowledged.

The VERIFY_ID object contains a node-unique value that is assigned by the generator of the BeginVerifyAck message. This value is used to uniquely identify the Verification process from multiple LMP neighbors and/or parallel Test procedures between the same LMP neighbors.

12.5.3. BeginVerifyNack Message (Msg Type = 7)

If a BeginVerify message is received and a node is unwilling or unable to begin the Verification procedure, a BeginVerifyNack message MUST be transmitted.

```
<BeginVerifyNack Message> ::= <Common Header> [<LOCAL_LINK_ID>]
                              <MESSAGE_ID_ACK> <ERROR_CODE>
```

The above transmission order SHOULD be followed.

The contents of the MESSAGE_ID_ACK object MUST be obtained from the BeginVerify message being negatively acknowledged.

If the Verification process is not supported, the ERROR_CODE MUST indicate "Link Verification Procedure not supported".

If Verification is supported, but the node is unable to begin the procedure, the ERROR_CODE MUST indicate "Unwilling to verify". If a BeginVerifyNack message is received with such an ERROR_CODE, the node that originated the BeginVerify SHOULD schedule a BeginVerify retransmission after Rf seconds, where Rf is a locally defined parameter.

If the Verification Transport mechanism is not supported, the ERROR_CODE MUST indicate "Unsupported verification transport mechanism".

If remote configuration of the Link_Id is not supported and the content of the REMOTE_LINK_ID object (included in the BeginVerify message) does not match any configured values, the ERROR_CODE MUST indicate "Link_Id configuration error".

If a node receives a BeginVerify message and recognizes the BEGIN_VERIFY object but does not recognize the C-Type, the ERROR_CODE MUST indicate "Unknown object C-Type".

12.5.4. EndVerify Message (Msg Type = 8)

The EndVerify message is sent over the control channel and is used to terminate the link verification process. The EndVerify message may be sent any time the initiating node desires to end the Verify procedure. The format is as follows:

```
<EndVerify Message> ::=<Common Header> <MESSAGE_ID> <VERIFY_ID>
```

The above transmission order SHOULD be followed.

The EndVerify message will be periodically transmitted until (1) an EndVerifyAck message has been received or (2) a retry limit has been reached and no EndVerifyAck message has been received. Both the retransmission interval and the retry limit are local configuration parameters.

12.5.5. EndVerifyAck Message (Msg Type =9)

The EndVerifyAck message is sent over the control channel and is used to acknowledge the termination of the link verification process. The format is as follows:

```
<EndVerifyAck Message> ::= <Common Header> <MESSAGE_ID_ACK>
                           <VERIFY_ID>
```

The above transmission order SHOULD be followed.

The contents of the MESSAGE_ID_ACK object MUST be obtained from the EndVerify message being acknowledged.

12.5.6. Test Message (Msg Type = 10)

The Test message is transmitted over the data link and is used to verify its physical connectivity. Unless explicitly stated, these messages MUST be transmitted over UDP like all other LMP messages. The format of the Test messages is as follows:

```
<Test Message> ::= <Common Header> <LOCAL_INTERFACE_ID> <VERIFY_ID>
```

The above transmission order SHOULD be followed.

Note that this message is sent over a data link and NOT over the control channel. The transport mechanism for the Test message is negotiated using the Verify Transport Mechanism field of the BEGIN_VERIFY object and the Verify Transport Response field of the BEGIN_VERIFY_ACK object (see Sections 13.8 and 13.9).

The local (transmitting) node sends a given Test message periodically (at least once every VerifyInterval ms) on the corresponding data link until (1) it receives a correlating TestStatusSuccess or TestStatusFailure message on the control channel from the remote (receiving) node or (2) all active control channels between the two nodes have failed. The remote node will send a given TestStatus message periodically over the control channel until it receives either a correlating TestStatusAck message or an EndVerify message.

12.5.7. TestStatusSuccess Message (Msg Type = 11)

The TestStatusSuccess message is transmitted over the control channel and is used to transmit the mapping between the local Interface_Id and the Interface_Id that was received in the Test message.

```
<TestStatusSuccess Message> ::= <Common Header> <LOCAL_LINK_ID>
                                <MESSAGE_ID> <LOCAL_INTERFACE_ID>
                                <REMOTE_INTERFACE_ID> <VERIFY_ID>
```

The above transmission order SHOULD be followed.

The contents of the REMOTE_INTERFACE_ID object MUST be obtained from the corresponding Test message being positively acknowledged.

12.5.8. TestStatusFailure Message (Msg Type = 12)

The TestStatusFailure message is transmitted over the control channel and is used to indicate that the Test message was not received.

```
<TestStatusFailure Message> ::= <Common Header> <MESSAGE_ID>
                                <VERIFY_ID>
```

The above transmission order SHOULD be followed.

12.5.9. TestStatusAck Message (Msg Type = 13)

The TestStatusAck message is used to acknowledge receipt of the TestStatusSuccess or TestStatusFailure messages.

```
<TestStatusAck Message> ::= <Common Header> <MESSAGE_ID_ACK>
                            <VERIFY_ID>
```

The above transmission order SHOULD be followed.

The contents of the MESSAGE_ID_ACK object MUST be obtained from the TestStatusSuccess or TestStatusFailure message being acknowledged.

12.6. Link Summary Messages

12.6.1. LinkSummary Message (Msg Type = 14)

The LinkSummary message is used to synchronize the Interface_Ids and correlate the properties of the TE link. The format of the LinkSummary message is as follows:

```
<LinkSummary Message> ::= <Common Header> <MESSAGE_ID> <TE_LINK>
                          <DATA_LINK> [<DATA_LINK>...]
```

The above transmission order SHOULD be followed.

The LinkSummary message can be exchanged any time a link is not in the Verification process. The LinkSummary message MUST be periodically transmitted until (1) the node receives a LinkSummaryAck or LinkSummaryNack message or (2) a retry limit has been reached and no LinkSummaryAck or LinkSummaryNack message has been received. Both the retransmission interval and the retry limit are local configuration parameters.

12.6.2. LinkSummaryAck Message (Msg Type = 15)

The LinkSummaryAck message is used to indicate agreement on the Interface_Id synchronization and acceptance/agreement on all the link parameters. It is on the reception of this message that the local node makes the Link_Id associations.

```
<LinkSummaryAck Message> ::=  <Common Header> <MESSAGE_ID_ACK>
```

The above transmission order SHOULD be followed.

12.6.3. LinkSummaryNack Message (Msg Type = 16)

The LinkSummaryNack message is used to indicate disagreement on non-negotiated parameters or propose other values for negotiable parameters. Parameters on which agreement was reached MUST NOT be included in the LinkSummaryNack message.

```
<LinkSummaryNack Message> ::= <Common Header> <MESSAGE_ID_ACK>
                              <ERROR_CODE> [<DATA_LINK>...]
```

The above transmission order SHOULD be followed.

The DATA_LINK objects MUST include acceptable values for all negotiable parameters. If the LinkSummaryNack includes DATA_LINK objects for non-negotiable parameters, they MUST be copied from the DATA_LINK objects received in the LinkSummary message.

If the LinkSummaryNack message is received and only includes negotiable parameters, then a new LinkSummary message SHOULD be sent. The values received in the new LinkSummary message SHOULD take into account the acceptable parameters included in the LinkSummaryNack message.

If the LinkSummary message is received with unacceptable, non-negotiable parameters, the ERROR_CODE MUST indicate "Unacceptable non-negotiable LINK_SUMMARY parameters."

If the LinkSummary message is received with unacceptable negotiable parameters, the ERROR_CODE MUST indicate "Renegotiate LINK_SUMMARY parameters."

If the LinkSummary message is received with an invalid TE_LINK object, the ERROR_CODE MUST indicate "Invalid TE_LINK object."

If the LinkSummary message is received with an invalid DATA_LINK object, the ERROR_CODE MUST indicate "Invalid DATA_LINK object."

If the LinkSummary message is received with a TE_LINK object but the C-Type is unknown, the ERROR_CODE MUST indicate, "Unknown TE_LINK object C-Type."

If the LinkSummary message is received with a DATA_LINK object but the C-Type is unknown, the ERROR_CODE MUST indicate, "Unknown DATA_LINK object C-Type."

12.7. Fault Management Messages

12.7.1. ChannelStatus Message (Msg Type = 17)

The ChannelStatus message is sent over the control channel and is used to notify an LMP neighbor of the status of a data link. A node that receives a ChannelStatus message MUST respond with a ChannelStatusAck message. The format is as follows:

```
<ChannelStatus Message> ::= <Common Header> <LOCAL_LINK_ID>
                            <MESSAGE_ID> <CHANNEL_STATUS>
```

The above transmission order SHOULD be followed.

If the CHANNEL_STATUS object does not include any Interface_Ids, then this indicates the entire TE Link has failed.

12.7.2. ChannelStatusAck Message (Msg Type = 18)

The ChannelStatusAck message is used to acknowledge receipt of the ChannelStatus Message. The format is as follows:

```
<ChannelStatusAck Message> ::= <Common Header> <MESSAGE_ID_ACK>
```

The above transmission order SHOULD be followed.

The contents of the MESSAGE_ID_ACK object MUST be obtained from the ChannelStatus message being acknowledged.

12.7.3. ChannelStatusRequest Message (Msg Type = 19)

The ChannelStatusRequest message is sent over the control channel and is used to request the status of one or more data link(s). A node that receives a ChannelStatusRequest message MUST respond with a ChannelStatusResponse message. The format is as follows:

```
<ChannelStatusRequest Message> ::= <Common Header> <LOCAL_LINK_ID>
                                   <MESSAGE_ID>
                                   [<CHANNEL_STATUS_REQUEST>]
```

The above transmission order SHOULD be followed.

If the CHANNEL_STATUS_REQUEST object is not included, then the ChannelStatusRequest is being used to request the status of ALL of the data link(s) of the TE Link.

12.7.4. ChannelStatusResponse Message (Msg Type = 20)

The ChannelStatusResponse message is used to acknowledge receipt of the ChannelStatusRequest Message and notify the LMP neighbor of the status of the data channel(s). The format is as follows:

```
<ChannelStatusResponse Message> ::= <Common Header> <MESSAGE_ID_ACK>
                                    <CHANNEL_STATUS>
```

The above transmission order SHOULD be followed.

The contents of the MESSAGE_ID_ACK objects MUST be obtained from the ChannelStatusRequest message being acknowledged.

13. LMP Object Definitions

13.1. CCID (Control Channel ID) Class

Class = 1

o C-Type = 1, LOCAL_CCID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            CC_Id                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

CC_Id: 32 bits

This MUST be node-wide unique and non-zero. The CC_Id identifies the control channel of the sender associated with the message.

This object is non-negotiable.

o C-Type = 2, REMOTE_CCID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             CC_Id                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

CC_Id: 32 bits

This identifies the remote node's CC_Id and MUST be non-zero.

This object is non-negotiable.

13.2. NODE_ID Class

Class = 2

o C-Type = 1, LOCAL_NODE_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Node_Id (4 bytes)                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Node_Id:

This identities the node that originated the LMP packet.

This object is non-negotiable.

o C-Type = 2, REMOTE_NODE_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Node_Id (4 bytes)                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Node_Id:

This identities the remote node.

This object is non-negotiable.

13.3. LINK_ID Class

Class = 3

o C-Type = 1, IPv4 LOCAL_LINK_ID

o C-Type = 2, IPv4 REMOTE_LINK_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Link_Id (4 bytes)                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

o C-Type = 3, IPv6 LOCAL_LINK_ID

o C-Type = 4, IPv6 REMOTE_LINK_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                        Link_Id (16 bytes)                     +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

o C-Type = 5, Unnumbered LOCAL_LINK_ID

o C-Type = 6, Unnumbered REMOTE_LINK_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Link_Id (4 bytes)                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Link_Id:

For LOCAL_LINK_ID, this identifies the sender's Link associated with the message. This value MUST be non-zero.

For REMOTE_LINK_ID, this identifies the remote node's Link_Id and MUST be non-zero.

This object is non-negotiable.

13.4. INTERFACE_ID Class

Class = 4

o C-Type = 1, IPv4 LOCAL_INTERFACE_ID

o C-Type = 2, IPv4 REMOTE_INTERFACE_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Interface_Id (4 bytes)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

o C-Type = 3, IPv6 LOCAL_INTERFACE_ID

o C-Type = 4, IPv6 REMOTE_INTERFACE_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                       Interface_Id (16 bytes)                 +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

o C-Type = 5, Unnumbered LOCAL_INTERFACE_ID

o C-Type = 6, Unnumbered REMOTE_INTERFACE_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Interface_Id (4 bytes)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Interface_Id:

For the LOCAL_INTERFACE_ID, this identifies the data link. This value MUST be node-wide unique and non-zero.

For the REMOTE_INTERFACE_ID, this identifies the remote node's data link. The Interface_Id MUST be non-zero.

This object is non-negotiable.

13.5. MESSAGE_ID Class

Class = 5

o C-Type=1, MessageId

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Message_Id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Message_Id:

The Message_Id field is used to identify a message. This value is incremented and only decreases when the value wraps. This is used for message acknowledgment.

This object is non-negotiable.

o C-Type = 2, MessageIdAck

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Message_Id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Message_Id:

The Message_Id field is used to identify the message being acknowledged. This value is copied from the MESSAGE_ID object of the message being acknowledged.

This object is non-negotiable.

13.6. CONFIG Class

Class = 6.

o C-Type = 1, HelloConfig

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         HelloInterval         |      HelloDeadInterval        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

HelloInterval: 16 bits.

Indicates how frequently the Hello packets will be sent and is measured in milliseconds (ms).

HelloDeadInterval: 16 bits.

If no Hello packets are received within the HelloDeadInterval, the control channel is assumed to have failed. The HelloDeadInterval is measured in milliseconds (ms). The HelloDeadInterval MUST be greater than the HelloInterval, and SHOULD be at least 3 times the value of HelloInterval.

If the fast keep-alive mechanism of LMP is not used, the HelloInterval and HelloDeadInterval MUST be set to zero.

13.7. HELLO Class

Class = 7

o C-Type = 1, Hello

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           TxSeqNum                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           RcvSeqNum                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TxSeqNum: 32 bits

This is the current sequence number for this Hello message. This sequence number will be incremented when the sequence number is reflected in the RcvSeqNum of a Hello packet that is received over the control channel.

TxSeqNum=0 is not allowed. TxSeqNum=1 is used to indicate that this is the first Hello message sent over the control channel.

RcvSeqNum: 32 bits

This is the sequence number of the last Hello message received over the control channel. RcvSeqNum=0 is used to indicate that a Hello message has not yet been received.

This object is non-negotiable.

13.8. BEGIN_VERIFY Class

Class = 8

o C-Type = 1

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Flags                      |         VerifyInterval        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Number of Data Links                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    EncType    |  (Reserved)   |  Verify Transport Mechanism   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        TransmissionRate                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Wavelength                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Reserved field should be sent as zero and ignored on receipt.

Flags: 16 bits

The following flags are defined:

0x0001 Verify all Links

If this bit is set, the verification process checks all unallocated links; else it only verifies new ports or component links that are to be added to this TE link.

0x0002 Data Link Type

If set, the data links to be verified are ports, otherwise they are component links

VerifyInterval: 16 bits

This is the interval between successive Test messages and is measured in milliseconds (ms).

Number of Data Links: 32 bits

This is the number of data links that will be verified.

EncType: 8 bits

This is the encoding type of the data link. The defined EncType values are consistent with the LSP Encoding Type values of [RFC3471].

Verify Transport Mechanism: 16 bits

This defines the transport mechanism for the Test Messages. The scope of this bit mask is restricted to each encoding type. The local node will set the bits corresponding to the various mechanisms it can support for transmitting LMP test messages. The receiver chooses the appropriate mechanism in the BeginVerifyAck message.

The following flag is defined across all Encoding Types. All other flags are dependent on the Encoding Type.

0x8000 Payload:Test Message transmitted in the payload

Capable of transmitting Test messages in the payload. The Test message is sent as an IP packet as defined above.

TransmissionRate: 32 bits

This is the transmission rate of the data link over which the Test messages will be transmitted. This is expressed in bytes per second and represented in IEEE floating-point format.

Wavelength: 32 bits

When a data link is assigned to a port or component link that is capable of transmitting multiple wavelengths (e.g., a fiber or waveband-capable port), it is essential to know which wavelength the test messages will be transmitted over. This value corresponds to the wavelength at which the Test messages will be transmitted over and has local significance. If there is no ambiguity as to the wavelength over which the message will be sent, then this value SHOULD be set to 0.

13.9. BEGIN_VERIFY_ACK Class

Class = 9

o C-Type = 1

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      VerifyDeadInterval       |   Verify_Transport_Response   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

VerifyDeadInterval: 16 bits

If a Test message is not detected within the VerifyDeadInterval, then a node will send the TestStatusFailure message for that data link.

Verify_Transport_Response: 16 bits

The recipient of the BeginVerify message (and the future recipient of the TEST messages) chooses the transport mechanism from the various types that are offered by the transmitter of the Test messages. One and only one bit MUST be set in the verification transport response.

This object is non-negotiable.

13.10. VERIFY_ID Class

Class = 10

o C-Type = 1

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Verify_Id                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Verify_Id: 32 bits

This is used to differentiate Test messages from different TE links and/or LMP peers. This is a node-unique value that is assigned by the recipient of the BeginVerify message.

This object is non-negotiable.

13.11. TE_LINK Class

Class = 11

o C-Type = 1, IPv4 TE_LINK

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                  (Reserved)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Local_Link_Id (4 bytes)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Remote_Link_Id (4 bytes)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

o   C-Type = 2, IPv6 TE_LINK

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                  (Reserved)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Local_Link_Id (16 bytes)                 +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Remote_Link_Id (16 bytes)                +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

o   C-Type = 3, Unnumbered TE_LINK

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                  (Reserved)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Local_Link_Id (4 bytes)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Remote_Link_Id (4 bytes)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Reserved field should be sent as zero and ignored on receipt.

Flags: 8 bits

The following flags are defined. All other bit-values are reserved and should be sent as zero and ignored on receipt.

0x01 Fault Management Supported.

0x02 Link Verification Supported.

Local_Link_Id:

This identifies the node’s local Link_Id and MUST be non-zero.

Remote_Link_Id:

This identifies the remote node’s Link_Id and MUST be non-zero.

13.12. DATA_LINK Class

Class = 12

o   C-Type = 1, IPv4 DATA_LINK

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                  (Reserved)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Local_Interface_Id (4 bytes)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Remote_Interface_Id (4 bytes)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
//                         (Subobjects)                        //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

o   C-Type = 2, IPv6 DATA_LINK

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                   (Reserved)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                   Local_Interface_Id (16 bytes)               +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                   Remote_Interface_Id (16 bytes)              +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
//                          (Subobjects)                       //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

o   C-Type = 3, Unnumbered DATA_LINK

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Flags     |                   (Reserved)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Local_Interface_Id (4 bytes)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Remote_Interface_Id (4 bytes)               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
//                          (Subobjects)                       //
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
   The Reserved field should be sent as zero and ignored on receipt.

   Flags: 8 bits

      The following flags are defined.  All other bit-values are
      reserved and should be sent as zero and ignored on receipt.

      0x01 Interface Type: If set, the data link is a port, otherwise it
                           is a component link.

      0x02 Allocated Link: If set, the data link is currently allocated
                           for user traffic.  If a single Interface_Id
                           is used for both the transmit and receive
                           data links, then this bit only applies to the
                           transmit interface.

      0x04 Failed Link:    If set, the data link is failed and not
                           suitable for user traffic.

   Local_Interface_Id:

      This is the local identifier of the data link.  This MUST be
      node-wide unique and non-zero.

   Remote_Interface_Id:

      This is the remote identifier of the data link.  This MUST be
      non-zero.

   Subobjects

      The contents of the DATA_LINK object consist of a series of
      variable-length data items called subobjects.  The subobjects are
      defined in Section 13.12.1 below.

   A DATA_LINK object may contain more than one subobject.  More than
   one subobject of the same Type may appear if multiple capabilities
   are supported over the data link.
```

13.12.1. Data Link Subobjects

The contents of the DATA_LINK object include a series of variable-length data items called subobjects. Each subobject has the form:

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---------------//--------------+
|    Type       |    Length     |      (Subobject contents)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+--------------//---------------+
```

Type: 8 bits

The Type indicates the type of contents of the subobject. Currently defined values are:

Type = 1, Interface Switching Type

Type = 2, Wavelength

Length: 8 bits

The Length contains the total length of the subobject in bytes, including the Type and Length fields. The Length MUST be at least 4, and MUST be a multiple of 4.

13.12.1.1. Subobject Type 1: Interface Switching Type

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type       |    Length     | Switching Type|   EncType     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Minimum Reservable Bandwidth                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Maximum Reservable Bandwidth                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Switching Type: 8 bits

This is used to identify the local Interface Switching Type of the TE link as defined in [RFC3471].

EncType: 8 bits

This is the encoding type of the data link. The defined EncType values are consistent with the LSP Encoding Type values of [RFC3471].

Minimum Reservable Bandwidth: 32 bits

This is measured in bytes per second and represented in IEEE floating point format.

Maximum Reservable Bandwidth: 32 bits

This is measured in bytes per second and represented in IEEE floating point format.

If the interface only supports a fixed rate, the minimum and maximum bandwidth fields are set to the same value.

13.12.1.2. Subobject Type 2: Wavelength

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type       |    Length     |         (Reserved)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Wavelength                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Reserved field should be sent as zero and ignored on receipt.

Wavelength: 32 bits

This value indicates the wavelength carried over the port. Values used in this field only have significance between two neighbors.

13.13. CHANNEL_STATUS Class

Class = 13

o C-Type = 1, IPv4 INTERFACE_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Interface_Id (4 bytes)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|A|D|                     Channel Status                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               :                               |
//                              :                              //
|                               :                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Interface_Id (4 bytes)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|A|D|                     Channel Status                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

o C-Type = 2, IPv6 INTERFACE_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Interface_Id (16 bytes)                  +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|A|D|                     Channel Status                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               :                               |
//                              :                              //
|                               :                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Interface_Id (16 bytes)                  +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|A|D|                     Channel Status                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

o C-Type = 3, Unnumbered INTERFACE_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Interface_Id (4 bytes)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|A|D|                     Channel Status                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               :                               |
//                              :                              //
|                               :                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Interface_Id (4 bytes)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|A|D|                     Channel_Status                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Active bit: 1 bit

This indicates that the Channel is allocated to user traffic and the data link should be actively monitored.

Direction bit: 1 bit

This indicates the direction (transmit/receive) of the data channel referred to in the CHANNEL_STATUS object. If set, this indicates the data channel is in the transmit direction.

Channel_Status: 30 bits

This indicates the status condition of a data channel. The following values are defined. All other values are reserved.

```
1   Signal Okay (OK):    Channel is operational
2   Signal Degrade (SD): A soft failure caused by a BER exceeding
                         a preselected threshold.  The specific
                         BER used to define the threshold is
                         configured.
3   Signal Fail (SF):    A hard signal failure including (but not
                         limited to) loss of signal (LOS), loss of
                         frame (LOF), or Line AIS.
```

This object contains one or more Interface_Ids followed by a Channel_Status field.

To indicate the status of the entire TE Link, there MUST be only one Interface_Id, and it MUST be zero.

This object is non-negotiable.

13.14. CHANNEL_STATUS_REQUEST Class

Class = 14

o C-Type = 1, IPv4 INTERFACE_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Interface_Id (4 bytes)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               :                               |
//                              :                              //
|                               :                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Interface_Id (4 bytes)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This object contains one or more Interface_Ids.

The Length of this object is 4 + 4N in bytes, where N is the number of Interface_Ids.

o C-Type = 2, IPv6 INTERFACE_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                       Interface_Id (16 bytes)                 +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               :                               |
//                              :                              //
|                               :                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                       Interface_Id (16 bytes)                 +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This object contains one or more Interface_Ids.

The Length of this object is 4 + 16N in bytes, where N is the number of Interface_Ids.

o C-Type = 3, Unnumbered INTERFACE_ID

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Interface_Id (4 bytes)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               :                               |
//                              :                              //
|                               :                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Interface_Id (4 bytes)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This object contains one or more Interface_Ids.

The Length of this object is 4 + 4N in bytes, where N is the number of of Interface_Ids.

This object is non-negotiable.

13.15. ERROR_CODE Class

Class = 20

o C-Type = 1, BEGIN_VERIFY_ERROR

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          ERROR CODE                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The following bit-values are defined in network byte order (i.e., big-endian byte order):

0x01 = Link Verification Procedure not supported.
0x02 = Unwilling to verify.
0x04 = Unsupported verification transport mechanism.
0x08 = Link_Id configuration error.
0x10 = Unknown object C-Type.

All other bit-values are reserved and should be sent as zero and ignored on receipt.

Multiple bits may be set to indicate multiple errors.

This object is non-negotiable.

If a BeginVerifyNack message is received with Error Code 2, the node that originated the BeginVerify SHOULD schedule a BeginVerify retransmission after Rf seconds, where Rf is a locally defined parameter.

o C-Type = 2, LINK_SUMMARY_ERROR

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          ERROR CODE                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The following bit-values are defined in network byte order (i.e., big-endian byte order):

0x01 = Unacceptable non-negotiable LINK_SUMMARY parameters.
0x02 = Renegotiate LINK_SUMMARY parameters.
0x04 = Invalid TE_LINK Object.
0x08 = Invalid DATA_LINK Object.
0x10 = Unknown TE_LINK object C-Type.
0x20 = Unknown DATA_LINK object C-Type.

All other bit-values are reserved and should be sent as zero and ignored on receipt.

Multiple bits may be set to indicate multiple errors.

This object is non-negotiable.

14. References

14.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4201] Kompella, K., Rekhter, Y., and L. Berger, "Link Bundling in MPLS Traffic Engineering (TE)", RFC 4201, October 2005.

[RFC4202] Kompella, K., Ed. and Y. Rekhter, Ed., "Routing Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS)", RFC 4202, October 2005.

[RFC2961] Berger, L., Gan, D., Swallow, G., Pan, P., Tommasi, F., and S. Molendini, "RSVP Refresh Overhead Reduction Extensions", RFC 2961, April 2001.

[RFC2402] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[RFC2406] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[RFC2407] Piper, D., "The Internet IP Security Domain of Interpretation for ISAKMP", RFC 2407, November 1998.

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[RFC3471] Berger, L., Ed., "Generalized MPLS - Signaling Functional Description", RFC 3471, January 2003.

14.2. Informative References

[RFC3630] Katz, D., Kompella, K., and D. Yeung, "Traffic Engineering (TE) Extensions to OSPF Version 2", RFC 3630, September 2003.

[RFC3784] Smit, H. and T. Li, "Intermediate System to Intermediate System (IS-IS) Extensions for Traffic Engineering (TE)", RFC 3784, June 2004.

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC3209] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, December 2001.

15. Security Considerations

There are number of attacks that an LMP protocol session can potentially experience. Some examples include:

- o an adversary may spoof control packets;
- o an adversary may modify the control packets in transit;
- o an adversary may replay control packets;
- o an adversary may study a number of control packets and try to break the key using cryptographic tools. If the hash/encryption algorithm used has known weaknesses, then it becomes easy for the adversary to discover the key using simple tools.

This section specifies an IPsec-based security mechanism for LMP.

15.1. Security Requirements

The following requirements are applied to the mechanism described in this section.

- o LMP security MUST be able to provide authentication, integrity, and replay protection.
- o For LMP traffic, confidentiality is not needed. Only authentication is needed to ensure that the control packets (packets sent along the LMP Control Channel) are originating from the right place and have not been modified in transit. LMP Test packets exchanged through the data links do not need to be protected.
- o For LMP traffic, protecting the identity of LMP end-points is not commonly required.
- o The security mechanism should provide for well defined key management schemes. The key management schemes should be well analyzed to be cryptographically secure. The key management schemes should be scalable. In addition, the key management system should be automatic.
- o The algorithms used for authentication MUST be cryptographically sound. Also, the security protocol MUST allow for negotiating and using different authentication algorithms.

15.2. Security Mechanisms

IPsec is a protocol suite that is used to secure communication at the network layer between two peers. This protocol is comprised of IP Security architecture document [RFC2401], IKE [RFC2409], IPsec AH [RFC2402], and IPsec ESP [RFC2406]. IKE is the key management protocol for IP networks, while AH and ESP are used to protect IP traffic. IKE is defined specific to IP domain of interpretation.

Considering the requirements described in Section 15.1, it is recommended that, where security is needed for LMP, implementations use IPsec as described below:

1. Implementations of LMP over IPsec protocol SHOULD support manual keying mode.

   Manual keying mode provides an easy way to set up and diagnose IPsec functionality.

   However, note that manual keying mode cannot effectively support features such as replay protection and automatic re-keying. An implementer using manual keys must be aware of these limits.

   It is recommended that an implementer use manual keying only for diagnostic purposes and use dynamic keying protocol to make use of features such as replay protection and automatic re-keying.

2. IPsec ESP with trailer authentication in tunnel mode MUST be supported.

3. Implementations MUST support authenticated key exchange protocols. IKE [RFC2409] MUST be used as the key exchange protocol if keys are dynamically negotiated between peers.

4. Implementation MUST use the IPsec DOI [RFC2407].

5. For IKE protocol, the identities of the SAs negotiated in Quick Mode represent the traffic that the peers agree to protect and are comprised of address space, protocol, and port information.

   For LMP over IPsec, it is recommended that the identity payload for Quick mode contain the following information:

   The identities MUST be of type IP addresses and the value of the identities SHOULD be the IP addresses of the communicating peers.

The protocol field MUST be UDP. The port field SHOULD be set to zero to indicate port fields should be ignored. This implies all UDP traffic between the peers must be sent through the IPsec tunnel. If an implementation supports port-based selectors, it can opt for a more finely grained selector by specifying the port field to the LMP port. If, however, the peer does not use port-based selectors, the implementation MUST fall back to using a port selector value of 0.

6. Aggressive mode of IKE negotiation MUST be supported.

When IPsec is configured to be used with a peer, all LMP messages are expected to be sent over the IPsec tunnel (crypto channel). Similarly, an LMP receiver configured to use Ipsec with a peer should reject any LMP traffic that does not come through the crypto channel.

The crypto channel can be pre-setup with the LMP neighbor, or the first LMP message sent to the peer can trigger the creation of the IPsec tunnel.

A set of control channels can share the same crypto channel. When LMP Hellos are used to monitor the status of the control channel, it is important to keep in mind that the keep-alive failure in a control channel may also be due to a failure in the crypto channel. The following method is recommended to ensure that an LMP communication path between two peers is working properly.

- o If LMP Hellos detect a failure on a control channel, switch to an alternate control channel and/or try to establish a new control channel.

- o Ensure the health of the control channels using LMP Hellos. If all control channels indicate a failure and it is not possible to bring up a new control channel, tear down all existing control channels. Also, tear down the crypto channel (both the IKE SA and IPsec SAs).

- o Reestablish the crypto channel. Failure to establish a crypto channel indicates a fatal failure for LMP communication.

- o Bring up the control channel. Failure to bring up the control channel indicates a fatal failure for LMP communication.

When LMP peers are dynamically discovered (particularly the initiator), the following points should be noted:

When using pre-shared key authentication in identity protection mode (main mode), the pre-shared key is required to compute the value of SKEYID (used for deriving keys to encrypt messages during key exchange). In main mode of IKE, the pre-shared key to be used has to be identified before receiving the peer's identity payload. The pre-shared key is required for calculating SKEYID. The only information available about the peer at this point is its IP address from which the negotiation came from. Keying off the IP address of a peer to get the pre-shared key is not possible since the addresses are dynamic and not known beforehand.

Aggressive mode key exchange can be used since identification payloads are sent in the first message.

Note, however, that aggressive mode is prone to passive denial of service attacks. Using a shared secret (group shared secret) among a number of peers is strongly discouraged because this opens up the solution to man-in-the-middle attacks.

Digital-signature-based authentication is not prone to such problems. It is RECOMMENDED that a digital-signature-based authentication mechanism be used where possible.

If pre-shared-key-based authentication is required, then aggressive mode SHOULD be used. IKE pre-shared authentication key values SHOULD be protected in a manner similar to the user's account password.

16. IANA Considerations

The IANA has assigned port number 701 to LMP.

In the following, guidelines are given for IANA assignment for each LMP name space. Ranges are specified for Private Use, to be assigned by Expert Review, and to be assigned by Standards Action (as defined in [RFC2434].

Assignments made from LMP number spaces set aside for Private Use (i.e., for proprietary extensions) need not be documented. Independent LMP implementations using the same Private Use code points will in general not interoperate, so care should be exercised in using these code points in a multi-vendor network.

Assignments made from LMP number spaces to be assigned by Expert Review are to be reviewed by an Expert designated by the IESG. The intent in this document is that code points from these ranges are used for Experimental extensions; as such, assignments MUST be accompanied by Experimental RFCs. If deployment suggests that these extensions are useful, then they should be described in Standards Track RFCs, and new code points from the Standards Action ranges MUST be assigned.

Assignments from LMP number spaces to be assigned by Standards Action MUST be documented by a Standards Track RFC, typically submitted to an IETF Working Group, but in any case following the usual IETF procedures for Proposed Standards.

The Reserved bits of the LMP Common Header should be allocated by Standards Action, pursuant to the policies outlined in [RFC2434].

LMP defines the following name spaces that require management:

- LMP Message Type.
- LMP Object Class.
- LMP Object Class type (C-Type). These are unique within the Object Class.
- LMP Sub-object Class type (Type). These are unique within the Object Class.

The LMP Message Type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-127 are allocated by Standards Action, 128-240 are allocated through an Expert Review, and 241-255 are reserved for Private Use.

The LMP Object Class name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range of 0-127 are allocated by Standards Action, 128-247 are allocated through an Expert Review, and 248-255 are reserved for Private Use.

The policy for allocating values out of the LMP Object Class name space is part of the definition of the specific Class instance. When a Class is defined, its definition must also include a description of the policy under which the Object Class names are allocated.

The policy for allocating values out of the LMP Sub-object Class name space is part of the definition of the specific Class instance. When a Class is defined, its definition must also include a description of the policy under which sub-objects are allocated.

The following name spaces have been assigned by IANA:

LMP Message Type name space

- o Config message (Message type = 1)
- o ConfigAck message (Message type = 2)
- o ConfigNack message (Message type = 3)
- o Hello message (Message type = 4)
- o BeginVerify message (Message type = 5)
- o BeginVerifyAck message (Message type = 6)
- o BeginVerifyNack message (Message type = 7)
- o EndVerify message (Message type = 8)
- o EndVerifyAck message (Message type = 9)
- o Test message (Message type = 10)
- o TestStatusSuccess message (Message type = 11)
- o TestStatusFailure message (Message type = 12)
- o TestStatusAck message (Message type = 13)
- o LinkSummary message (Message type = 14)
- o LinkSummaryAck message (Message type = 15)
- o LinkSummaryNack message (Message type = 16)
- o ChannelStatus message (Message type = 17)
- o ChannelStatusAck message (Message type = 18)
- o ChannelStatusRequest message (Message type = 19)
- o ChannelStatusResponse message (Message type = 20)

LMP Object Class name space and Class type (C-Type)

o CCID Class name (1)

The CCID Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

- LOCAL_CCID (C-Type = 1)
- REMOTE_CCID (C-Type = 2)

o NODE_ID Class name (2)

The NODE ID Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

- LOCAL_NODE_ID (C-Type = 1)
- REMOTE_NODE_ID (C-Type = 2)

o LINK_ID Class name (3)

The LINK_ID Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

- IPv4 LOCAL_LINK_ID (C-Type = 1)
- IPv4 REMOTE_LINK_ID (C-Type = 2)
- IPv6 LOCAL_LINK_ID (C-Type = 3)
- IPv6 REMOTE_LINK_ID (C-Type = 4)
- Unnumbered LOCAL_LINK_ID (C-Type = 5)
- Unnumbered REMOTE_LINK_ID (C-Type = 6)

o INTERFACE_ID Class name (4)

The INTERFACE_ID Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

```
  - IPv4 LOCAL_INTERFACE_ID         (C-Type = 1)
  - IPv4 REMOTE_INTERFACE_ID        (C-Type = 2)
  - IPv6 LOCAL_INTERFACE_ID         (C-Type = 3)
  - IPv6 REMOTE_INTERFACE_ID        (C-Type = 4)
  - Unnumbered LOCAL_INTERFACE_ID   (C-Type = 5)
  - Unnumbered REMOTE_INTERFACE_ID  (C-Type = 6)
```

o MESSAGE_ID Class name (5)

The MESSAGE_ID Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

```
  - MESSAGE_ID                      (C-Type = 1)
  - MESSAGE_ID_ACK                  (C-Type = 2)
```

o CONFIG Class name (6)

The CONFIG Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

```
  - HELLO_CONFIG                    (C-Type = 1)
```

o HELLO Class name (7)

The HELLO Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

```
  - HELLO                           (C-Type = 1)
```

o BEGIN_VERIFY Class name (8)

The BEGIN_VERIFY Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

```
  - Type 1                          (C-Type = 1)
```

o BEGIN_VERIFY_ACK Class name (9)

The BEGIN_VERIFY_ACK Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

- Type 1 (C-Type = 1)

o VERIFY_ID Class name (10)

The VERIFY_ID Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

- Type 1 (C-Type = 1)

o TE_LINK Class name (11)

The TE_LINK Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

- IPv4 TE_LINK (C-Type = 1)
- IPv6 TE_LINK (C-Type = 2)
- Unnumbered TE_LINK (C-Type = 3)

o DATA_LINK Class name (12)

The DATA_LINK Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for private Use.

- IPv4 DATA_LINK (C-Type = 1)
- IPv6 DATA_LINK (C-Type = 2)
- Unnumbered DATA_LINK (C-Type = 3)

The DATA_LINK Sub-object Class name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range of 0-127 are allocated by Standards Action, 128-247 are allocated through an Expert Review, and 248-255 are reserved for private Use.

- Interface Switching Type (sub-object Type = 1)
- Wavelength (sub-object Type = 2)

o CHANNEL_STATUS Class name (13)

The CHANNEL_STATUS Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

- IPv4 INTERFACE_ID (C-Type = 1)
- IPv6 INTERFACE_ID (C-Type = 2)
- Unnumbered INTERFACE_ID (C-Type = 3)

o CHANNEL_STATUS_REQUESTClass name (14)

The CHANNEL_STATUS_REQUEST Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for Private Use.

- IPv4 INTERFACE_ID (C-Type = 1)
- IPv6 INTERFACE_ID (C-Type = 2)
- Unnumbered INTERFACE_ID (C-Type = 3)

o ERROR_CODE Class name (20)

The ERROR_CODE Object Class type name space should be allocated as follows: pursuant to the policies outlined in [RFC2434], the numbers in the range 0-111 are allocated by Standards Action, 112-119 are allocated through an Expert Review, and 120-127 are reserved for private Use.

- BEGIN_VERIFY_ERROR (C-Type = 1)
- LINK_SUMMARY_ERROR (C-Type = 2)

17. Acknowledgements

The authors would like to thank Andre Fredette for his many contributions to this document. We would also like to thank Ayan Banerjee, George Swallow, Adrian Farrel, Dimitri Papadimitriou, Vinay Ravuri, and David Drysdale for their insightful comments and suggestions. We would also like to thank John Yu, Suresh Katukam, and Greg Bernstein for their helpful suggestions for the in-band control channel applicability.

18. Contributors

Jonathan P. Lang
Sonos, Inc.
223 E. De La Guerra St.
Santa Barbara, CA 93101

EMail: jplang@ieee.org

Krishna Mitra
Independent Consultant

EMail: kmitra@earthlink.net

John Drake
Calient Networks
5853 Rue Ferrari
San Jose, CA 95138

EMail: jdrake@calient.net

Kireeti Kompella
Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, CA 94089

EMail: kireeti@juniper.net

Yakov Rekhter
Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, CA 94089

EMail: yakov@juniper.net

Lou Berger
Movaz Networks

EMail: lberger@movaz.com

Debanjan Saha
IBM Watson Research Center

EMail: dsaha@us.ibm.com

Debashis Basak
Accelight Networks
70 Abele Road, Suite 1201
Bridgeville, PA 15017-3470

EMail: dbasak@accelight.com

Hal Sandick
Shepard M.S.
2401 Dakota Street
Durham, NC 27705

EMail: sandick@nc.rr.com

Alex Zinin
Alcatel

EMail: alex.zinin@alcatel.com

Bala Rajagopalan
Intel Corp.
2111 NE 25th Ave
Hillsboro, OR 97123

EMail: bala.rajagopalan@intel.com

Sankar Ramamoorthi
Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, CA 94089

EMail: sankarr@juniper.net

Contact Address

Jonathan P. Lang
Sonos, Inc.
829 De La Vina, Suite 220
Santa Barbara, CA 93101

EMail: jplang@ieee.org

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.